Efficient Algorithms for the Maximum Convex Sum Problem

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science in Computer Science and Software Engineering

in the

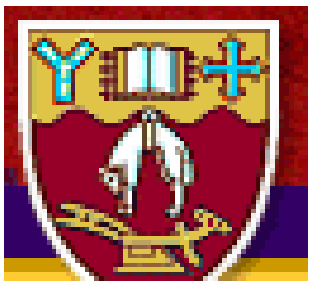
University of Canterbury

by

Mohammed Thaher

2009

Associate Prof. Dr. Alan Sprague, University of Alabama	External Examiner
Prof. Tadao Takaoka, University of Canterbury	Supervisor
Dr. R. Mukundan, University of Canterbury	Co-Supervisor



5th February 2009

This thesis is dedicated to

My parents, lovely wife and my daughters for their

endless love and constant support.

Abstract

The work of this thesis covers the Maximum Subarray Problem (MSP) from a new perspective. Research done previously and current methods of finding MSP include using the rectangular shape for finding the maximum sum or gain. The rectangular shape region used previously is not flexible enough to cover various data distributions. This research suggested using the convex shape, which is expected to have optimised and efficient results.

The steps to build towards using the proposed convex shape in the context of MSP are as follows: studying the available research in-depth to extract the potential guidelines for this thesis research; implementing an appropriate convex shape algorithm; generalising the main algorithm (based on dynamic programming) to find the second maximum sum, the third maximum sum and up to K^{th} maximum sum; and finally conducting experiments to evaluate the outcomes of the algorithms in terms of the maximum gain, time complexity, and the running time.

In this research, the following findings were achieved: one of the achievements is presenting an efficient algorithm, which determines the boundaries of the convex shape while having the same time complexity as other existing algorithms (the prefix sum was used to speed up the convex shape algorithm in finding the maximum sum). Besides the first achievement, the algorithm was generalized to find up to the K^{th} maximum sum. Finding the K^{th} maximum convex sum was shown to be useful in many applications, one of these (based on a study with the cooperation of

Christchurch Hospital in New Zealand) is accurately and efficiently locating brain tumours. Beside this application, the research findings present new approaches to applying MSP algorithms in real life applications, such as data mining, computer vision, astronomy, economics, chemistry, and medicine.

Acknowledgments

The work on this thesis has been an inspiring, often exciting, sometimes challenging, but always interesting experience. It has been made possible by many other people, who have supported me.

I would like to express thanks to my supervisor, Professor Tadao Takaoka, for his time and patient guidance, encouragement and excellent advice throughout this study.

Special thanks for Sung Eun Bae's valuable research. Many Thanks also to the following people: Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, Takeshi Tokuyama, and Christchurch hospital Oncology department staff.

I am deeply and forever indebted to my parents for their love, support and encouragement throughout my entire life.

I take this opportunity to express my profound gratitude to my beloved wife expressing my deepest gratitude for her constant support, understanding and love that I have received. I would like to express my thanks to my parents in law for their support and encouragement. Last, but not least, I would like to thank my lovely

daughters for the cheerful moments during my study; their smiles keep me going forward.

Finally, I would like to thank those other people whom may have contributed to the successful realization of the thesis.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

1.1 Overview

The Maximum Subarray Problem (MSP) is the problem of finding the most useful and informative array portion that associates two parameters involved in data. The MSP is considered to be an efficient data mining method, which gives an accurate trend of data with respect to some associated parameters.

An introductory detailed example of MSP is given in this section, to illustrate the different concepts and to highlight the involved theory. This example is the record of a bank customer's annual balance and the customer's age. These two parameters can be used to optimise the selection process in offering credit cards to the "right" customers. This data can be analysed by using matrices (for example, a two-dimensional array), where the input data is the quantity of transactions in association with two parameters age (rows) and annual balance (columns) – Figure 1.1. The output is the portion of the original matrix that corresponds to the value of the required (maximum) output value, which an algorithm is required to extract. In our case this represents the most promising range of customers' credit cards.

The actual formulation of the problem is as follows:

Suppose the record of transactions of bank customer's is given in the following two-dimensional input array of Figure 1.1.

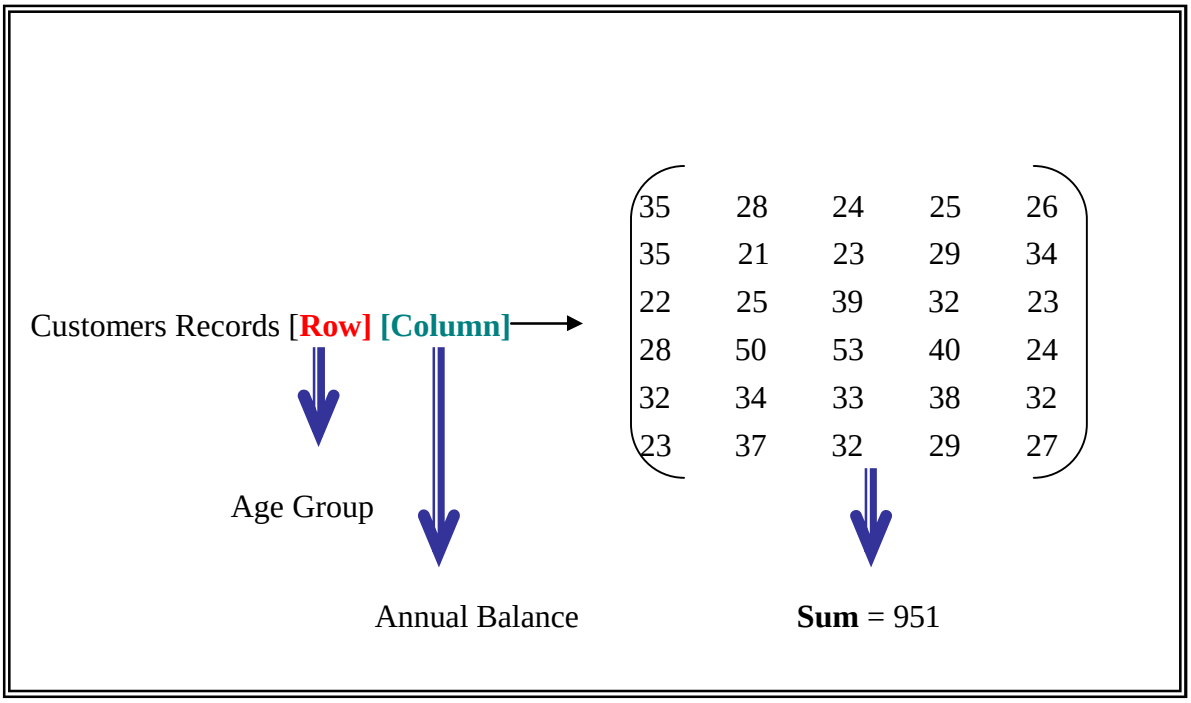


Figure 1.1: Input array extracted from the bank's database

If we consider the given input array with all positive numbers, the interval solution for the MSP is the sum of the whole array, which is 951 in the above example. The values assigned to each number of the two-dimensional array of the main database of the records are non-negative; this implies that the maximum subarray is the whole array, which is a trivial solution and of no interest. To overcome this, the values of the elements are normalized, before computing the maximum subarray, by subtracting a positive anchor value, which can be the overall mean. After applying this step, the maximum subarray can be computed, which corresponds to offering particular customers credit cards.

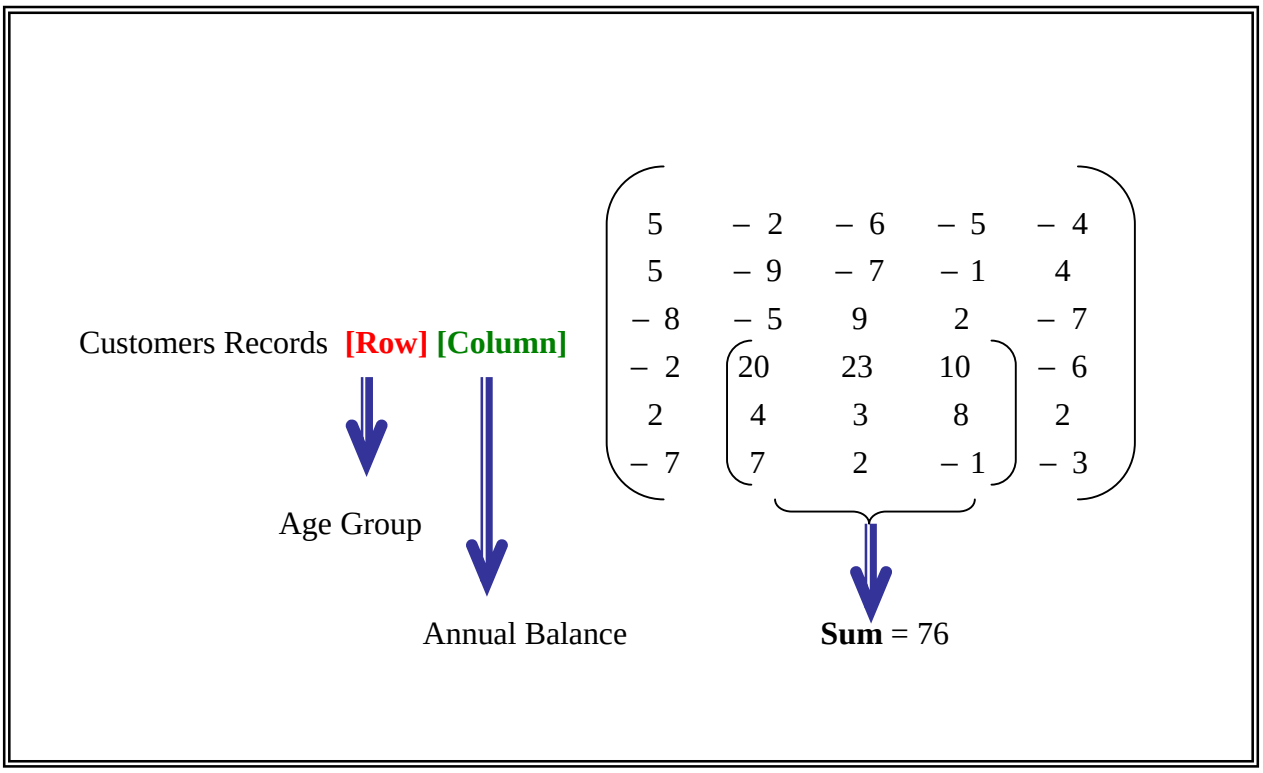


Figure 1.2: The input matrix has a portion that can be considered to be the maximum subarray, which can be useful for bank marketing purposes

The maximum subarray problem in Figure 1.2 is determined by an algorithm that uses a particular shape, which is in this example a rectangular shape/portion of the matrix. One can track the position of an element in the array by following its index. For example (4, 2) corresponds to the upper left corner and (6, 4) corresponds to the lower right corner. Using such a method for maximum subarray mining would result in finding the range of age groups and annual balance levels that would return the best revenue for the bank and would be a beneficial contribution in planning efficient marketing strategies.

The above example can be extended to include the K maximum subarray problem (K-MSP) to be able to offer a wider range of the bank customers the services that would

further increase the bank revenue. The goal of K-MSP is to find K maximum subarrays. K is a positive number, which is between 1 and $\frac{mn}{4}(m+1)(n+1)$, where m and n refer to the size of the given array. There are two techniques to find K-MSP; the disjoint method and the overlapping method. In the disjoint case, all maximum subarrays that have been detected in a given array must be separated or disjoint from each other, whereas, in the overlapping case, we are not enforced by such restriction. These methods can be applied to the above example.

Part of the bank marketing strategy that can be applied in the aforementioned example is to find the second targeted group of customers, who fall just below the threshold when determining the first targeted ones by using MSP. These customers can be encouraged by certain methods, for example sending flyers, to lead them getting a credit card, while returning the best revenue to the bank. Another category of the bank customers can be covered by finding the third Maximum sum. These customers can be approached by phone calls or person to person meetings. The fourth maximum category can be encouraged by giving out gifts. Ideas on the following maximum sums can be applied, to achieve optimized revenue.

1.2 History of maximum subarray problem

The maximum subarray problem emerged as a result of encountering a problem in pattern recognition in 1977 by Grenander at Brown University. Grenander needed to find the maximum sum over all of the rectangular regions of a given m × n array of real numbers. The maximum likelihood estimator of a certain kind of pattern in a digitized picture was to be represented by the maximum sum or maximum subarray.

For an array of size n×n, Grenander implemented an O(n^6) time algorithm. This algorithm was considered to be slow. In attempts to reduce the time factor, he simplified the problem to one dimension (1D), in order to gain an understanding of the structure. The input was a one dimensional-array of m real numbers, and the output was the maximum sum obtained in any neighboring subarray of the input.

Grenander managed to obtain O(n^3) time using one-dimensional array. Following this, Shamos and Bentley improved the complexity to O(n^2) and later implemented an O(n log n) time algorithm[1]. This result was changed two weeks later, when Kadane suggested a solution, while attending a seminar by Shamos [1], which has a linear time algorithm. While attending a seminar by Bentley, Gries also suggested a similar linear time algorithm [1].

As a result of the computational expense of all known algorithms, Grenander gave up on attempting to use the maximum subarray to solve the problem of pattern matching. The two-dimensional (2D) version of this problem was found to be solved in O (n^3) time by extending Kadane's algorithm [3]. Smith also presented O (n) time for the one-dimension and O(n^3) time for the two-dimension based on divide-and-conquer techniques [1].

Currently, the optimal time for the 1D version is O (n) time. While O(n^3) time for the two-dimensional MSP was considered to be the best time until Tamaki and Tokuyama devised an algorithm achieving sub-cubic time of O(n^3 (log log n/ log n) $^{1/2}$) [1] by adopting a divide-and-conquer technique and applying the fastest known Distance

Matrix Multiplication (DMM) algorithm by Takaoka [20]. Recently, Takaoka simplified the algorithm [12] and later on presented even faster DMM algorithm that is directly related to MSP applications [20].

In 2007, Bashar and Takaoka [20], developed efficient algorithms for K-MSP the time complexity for K-MSP became $O(Kn^2 \log^2 n \log K)$ when $K \leq n/\log n$ based on K-Tuple Approach.. Also, they conducted the average case analysis of algorithms for MSP and K-MSP. Furthermore, they compared and evaluated K-MSP. In the same year, Bae's research identified two categories of K-maximum subarray problem, the K-overlapping maximum subarray problem (K-OMSP) and the K-disjoint maximum subarray problem (K-DMSP) [1]. Bae presented various techniques to speed up computing these problems. Also, he adopted the general framework based on the subtraction of minimum prefix sum from each prefix sum to produce candidates for the final solution. Furthermore, he investigated various methods to compute the K-OMSP efficiently.

In addition to the above-mentioned research on MSP, Fukuda and Morimoto published a paper that discusses data mining based on association rules for two numeric attributes and one Boolean attribute [4]. They proposed an efficient algorithm for computing the regions that give optimal association rules for gain, support and confidence. The main aim of the Fukuda and Morimoto algorithm was to generate two-dimensional association rules that represent the dependence of the probability that an object condition will be met on a pair of numeric attributes. Alan Sprague in a

valuable research investigated extracting optimal association rules over numerical attributes by using the anchored convex shape [25]

1.3 Applications of MSP

There are many applications of MSP. Three of these are computer vision, data mining and a medical application in Oncology (Breast Cancer). An example of each application is discussed in the following subsections.

1.3.1 Computer vision

MSP can be used to find the brightest portion inside an image in computer vision (Figure 1.3). Following a particular colour scheme/scale, a score can be assigned to the required spot in the image. A two-dimensional array can be used to represent a bitmap image, where each cell/pixel represents the colour value based on RGB standard. Brightness is defined as (R+G+B)/3. The two-dimensional array that represents the image can store the value of brightness of each pixel in the corresponding location of the bitmap image.

Figure 1.3: The rectangular area shown in the figure can be obtained by running an algorithm in two-dimensional array MSP on the bitmap images.

It is known that the above definition of brightness does not correspond well to human colour perception [1]. The three television standards (NTSC, PAL, SECAM) use

luminance for most graphical applications. Luminance (Y) can be obtained by using the following equation:

$$Y = 0.30R + 0.59G + 0.11B.$$

Here, weightings 0.30, 0.59, and 0.11 are chosen to closely match the sensitivity of the eye to red, green, and blue [1].

The values assigned to each cell of the two-dimensional array of the image are non-negative; this implies that the maximum subarray is the whole array, which is a trivial solution and of no interest. To overcome this, the values of the cell elements are normalized. After applying this step, the maximum subarray can be computed, which corresponds to the brightest area in the image.

One of the major challenges for the MSP-based graphical applications is computational efficiency. Although the upper bound for the two-dimensional array MSP has been reduced to sub-cubic [1], it is still close to cubic time and performing pixel-wise operation by a near-cubic time algorithm can be time consuming.

1.3.2 Data mining

An example is given in this subsection to illustrate how MSP can be involved in data mining [3]. Suppose that a relationship is identified between item X and item Y, where X is an item that a customer buys from a store and Y is a potential item that the customer may buy if he/she buys X item (Tables 1 and 2) [3]. Items X and Y are associated with an association rule which is denoted by X g Y. This is measured by the formula of confidence, such that,

conf(X g Y) = support(X, Y) / support(X) .

Where X is the antecedent and Y the consequence. Support (X, Y) is the number of transactions that include both X and Y, and support (X) is that for X alone.

Customer Number	Items	Expenditure
1	beef, butter, cereal, milk	$41
2	bread, butter, milk	$23
3	beef, bread, butter, milk	$36
4	bread, milk	$14
5	bread, cereal, milk	$26
6	beef, bread, butter, cereal	$45

Table 1: Customers' purchase record

	Customer Name	Gender	Age	Annual income	Address
1	Lisa	F	36	$20,000	suburb B
2	Alison	F	47	$35,000	suburb B
3	John	M	27	$25,000	suburb A
4	Andrew	M	50	$60,000	suburb A
5	Michel	M	62	$65,000	suburb B
6	Linda	F	38	$45,000	suburb A

Table 2: Customers' personal record

In this example, a supermarket is recording the customers' purchases. Table 1 shows each customer's purchase transactions and Table 2 has a record of customers' personal data. Using these tables, a relationship between "butter" and "beef" can be found, where three customers who bought "butter" also bought "beef". This implies that conf (beef g butter) = 3/3 = 1.

Besides the above relationship, a relationship between a particular age group and a certain purchase can also be found or a link between income level and purchases can be investigated, where the same principles can be applied to discover rules with numerical attributes. Srikant and Agrawal [8] provide detailed research in this area.

Referring to the example of data mining of finding a certain product that is likely to be bought by a particular age group, for example, “beef”. We have two numerical attributes of customers: their age and their purchase amount for “beef”. In the example, if the age range is set to age ≤ 50, the confidence conf(age ≤ 50 g beef) is found to be 3/5. However, if the range age is set to age < 40 for the antecedent, the confidence conf(age < 40 g beef) is found to be 1. This finding indicates that a cost-efficient advertising outcome can be achieved, if the advertisement for “beef” is targeted at customers younger than 40.

If we assume that an array *a* of n numbers is used, the above example can be formalised by the MSP. This array can be a one-dimensional array of size 6, where *a*[i] represents the number of customers whose ages are as follows: 10i ≤ age <10(i + 1), and bought “beef”. According to the above tables (Table 1 and 2), the array “*a*” could be *a* = {0, 1, 2, 0, 0, 0}; this is normalized by subtracting the mean 1/2, such that *a* = {-1/2, 1/2, 3/2, -1/2, -1/2, -1/2}. The maximum subarray of the array was found in the portion *a*[2,3]. This indicates that customers in their twenties and thirties are most likely to purchase “beef”.

The above example indicates how MSP is applied on one-dimensional array. If two numerical attributes are used for the antecedent, the problem is then equivalent to the two-dimensional MSP. Maximum subarray problem in the context of two-dimensional association rules was described in Fukuda and Morimoto's research (1994) [4].

1.3.3 Medical application in Oncology

Breast Cancer is a common disease in women worldwide. As an example, in New Zealand, the current statistics show one in three cancers is breast cancer. Approximately 2,500 women are diagnosed with breast cancer each year. Over six women are diagnosed with breast cancer each day. Over six hundred women die from breast cancer each year; one in ten women in New Zealand will be diagnosed with breast cancer in their lifetime. More than 70% of women diagnosed with breast cancer will survive [22-24].

Cancer is a disease that attacks body cells (Figures 1.4 and 1.5). Bodies continually make new cells to replace worn-out cells or to heal damaged cells after an injury. Certain genes control this process; cancers are caused by damage to these genes. [24].

Normally, cells grow and multiply in an orderly way. However, damaged genes can cause them to behave abnormally. They may grow into a lump, which is called a tumour. Tumours can be benign (not cancerous) or malignant (cancerous). Benign lumps do not spread to other parts of the body, but have to be treated as they have effects on the local area. A malignant tumour, when it first develops, may be confined to its original site, however if these cells are not treated they may spread into

surrounding tissue and other parts of the body. To be able to offer appropriate treatment, the lump(s) have to be located.

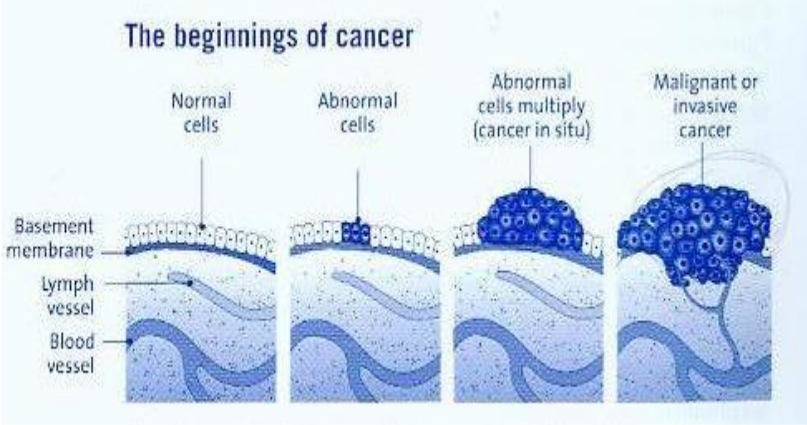


Figure 1.4: The first stages of cancer cells growth

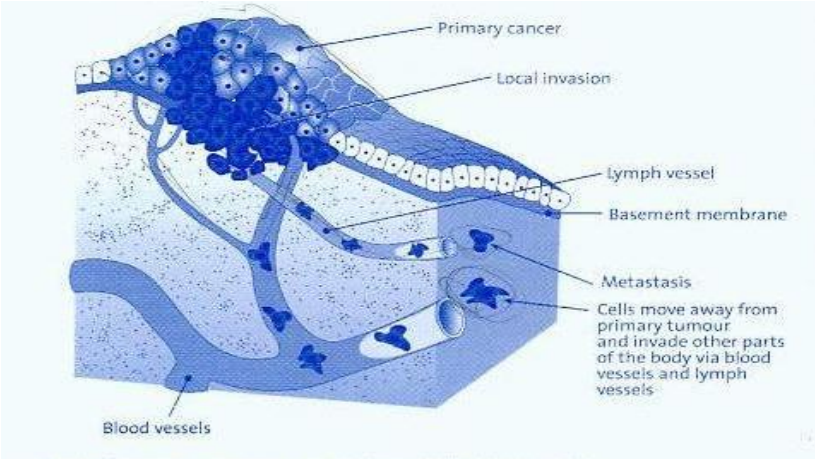


Figure 1.5: The shape of the formed cancer lump (cells) at an early stage

Accurately locating the tumour is not a straight forward task. Research from this thesis can aid in using an appropriate algorithm to locate the tumour at early stages. It was found that the shape of the developed tumour in breast cancer at the beginning comes under a family of shapes called convex shapes. Also, it was noted that the tumour shape has similar characteristics to a shape known as the WN convex shape.

This shape will be covered in details in Chapter 3. The common characteristics imply that applying the concept of the maximum sum can result in accurately locating the lump(s).

1.4 Research scope

Previous studies and applications used a rectangular shape region (Figure 1.6) to find the maximum subarray for a particular problem. In this research, a convex shape is suggested (Figure 1.7) to be used in MSP. The convex shape is more likely to result in relatively optimised results. The rectangular shape region used previously is not flexible enough to cover various data distributions;

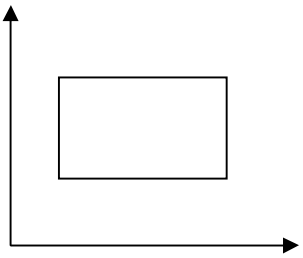


Figure 1.6: A rectangular region used to find maximum subarray

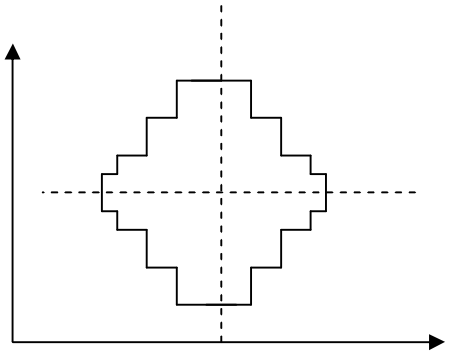


Figure 1.7: A convex region that can be used to find the maximum sum or gain in MSP

1.5 Research objectives

This subsection briefly highlights the major proposed research objectives. There are three main research goals in this thesis. The first goal is to find an area using the convex shape to maximize the sum or gain in MSP. The second goal is to compare (using random numbers) the maximum sum obtained by using the rectangular shape in MSP with that obtained by using the convex shape. The third goal, which is the most challenging one, is to generalize the convex algorithm up to the k^{th} maximum problem based on dynamic programming.

1.6 Thesis structure

Chapter 2 of this research presents the theoretical foundation of MSP. Chapter 3 covers the convex shape in the context of finding optimised maximum sum. Chapter 4 presents results, analysis and discussion of the evaluation experiments. Chapter 5 highlights the conclusions of this research and future work.

Chapter 2

Theoretical Foundation

This chapter presents an overview of maximum subarray problem. This overview includes the new approach to dealing with MSP- the convex shape. Definitions, theoretical foundation and examples of MSP will be covered. These include the maximum subarray problem in one dimension and maximum subarray problem in two dimensions.

2.1 Maximum Subarray Problem

2. 1.1 Maximum subarray in one-dimensional array

2.1.1.1 Problem definition

MSP can be simply defined as finding the contiguous array portion, called a subarray, within a one-dimensional array of numbers that has the maximum sum. For example, for the sequence of values -5, 2, -3, 4, -1, 4, 2, -5, 4, the subarray with the greatest sum is 4, -1, 4, 2, with sum 9.

MSP in a one-dimensional array can be formally defined as follows: if we assume that MAX(E) is the operation that selects the maximum element in a list E. For a given

array a[1..n] containing positive and negative real numbers and 0, the maximum subarray is the consecutive array elements of the greatest sum, such that,

$$\text{M} = \text{MAX(E)}, \text{ where E} = \left\{ \sum_{x=i}^{j} a[x] \,|\, 1 \le i \le j \le n \right\}$$

There are several algorithms to compute the maximum subarray in one dimension. In this study, Kadane's Algorithm, prefix sum algorithm and Maximum Sum in a one-dimensional array will be discussed.

First, the linear time algorithm for one-dimensional array, which is known as Kadane's algorithm [1], will be covered. This algorithm is designed to find the portion from a one-dimensional array that gives the maximum sum. Algorithm 1 shows how this can be computed.

2.1.1.1.1 Kadane's algorithm for one-dimensional array

Using Kadane's algorithm in one-dimension, a tentative sum in *h* is accumulated by scanning the array. If $h > p$ for the current maximum *p*, *p* is updated to *h* . The value of *h* is reset to zero if it becomes negative. In this algorithm, the variables *m* and *k* keep track of the beginning and ending positions of the subarray whose sum is *p*. Figure 2.1 illustrates the situation.

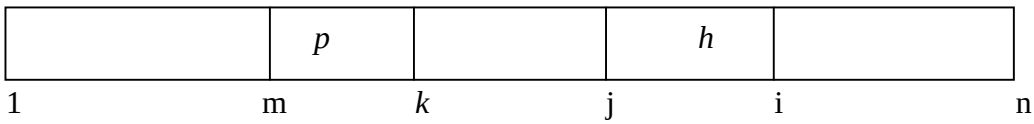


Figure 2.1: An example to illustrate Kadane's algorithm for one-dimensional array

If all the values are negative, we allow the empty subarray with $p = 0$. In this case, $(m,k) = (0,0)$ will not change.

Algorithm (1)
Kadane's one-dimensional array algorithm O(n)

1. (m,l) = (0,0) ; p = 0; h = 0; j = 1;
2. for (i=1; i<=n ; i++){
h=h +a[i];
if (h>p) { (k,l) = (j,i); p=h;}
if (h<0) then { h= 0 ;j=i+1;}
3. }

The computing time for the above algorithm is O(n), that is, linear time; this is the optimal solution for this problem.

2. 1.1.1.1.A Algorithm for one-dimensional array by prefix sum

Given array l at position i, the prefix sum of this array, denoted by sum[i], is the sum of l [1],……., l [i]. The prefix sum array *sum* is computed in O (n) time as follows:

Algorithm (2)
Prefix-Sum algorithm O(n)

sum[0]=0;
for(i=1; i<=n;i++)
sum[i]= sum[i-1]+ l [i];

The following is an example of the prefix sum, where each element is obtained from the sum of preceding elements:

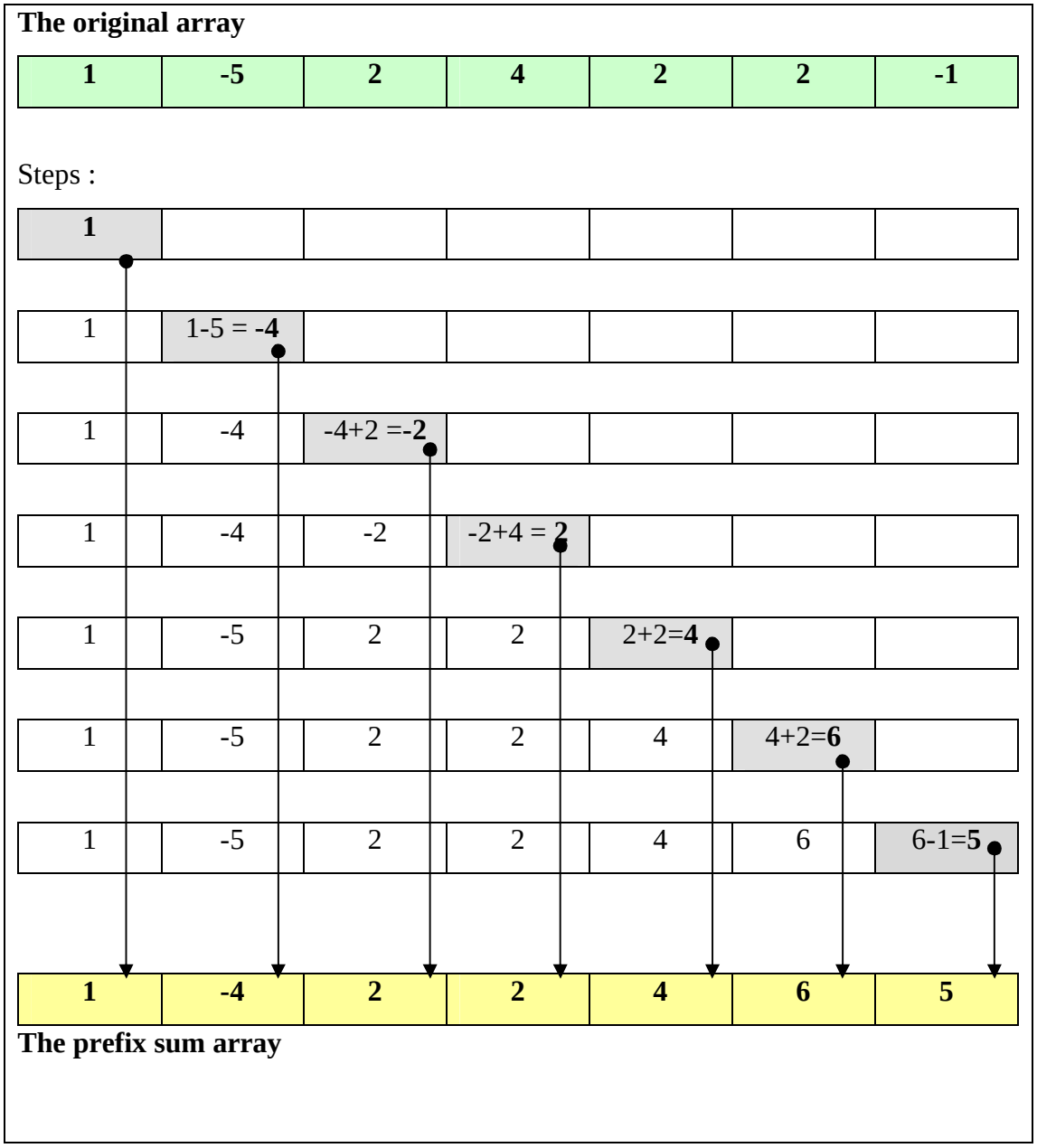


Figure 2.2: An example to illustrate the prefix sum in one-dimensional array

If sum[s] = $\sum_{i=1}^{t} I[i]$, the sum of I [s..t] is computed by subtraction of these prefix sums such as:

$$\sum_{i=s}^{t} I[i] = \text{sum}[t] - \text{sum}[s-1]$$

To yield the maximum sum from one-dimension array, we have to find indices s, t that maximize $\sum_{i=s}^{t} I[i]$. Let min_i be the minimum prefix sum for an array portion I [1..i-1]. Then the following lemma is obvious.

2.1.1.1.1.B Lemma 1*:*

For all s, t • [1..n] and s ≤ t,

$$\underset{1\le s\le t\le n}{\text{MAX}} \left\{ \sum_{i=s}^{t} I[i] \right\} = \underset{1\le s\le t\le n}{\text{MAX}} \{\text{sum[t]- sum[s-1]}\}$$

$$= \underset{1\le t\le n}{\text{MAX}} \left\{ \text{sum[t]} - \underset{1\le s\le t}{MIN} \{\text{sum[s-1]}\} \right\}$$

$$= \underset{1\le t\le n}{\text{MAX}} \left\{ \text{sum[t]-min}_t \right\}$$

Based on *Lemma* 1, we can devise a linear time algorithm (3) that finds the maximum sum in a one-dimensional array.

Algorithm (3)
Maximum sum in a one-dimensional array
1: min = 0 //minimum prefix sum 2: M = 0 //current solution. 0 for empty subarray 3: sum[0] = 0 4: for i = 1 to n do { sum[i] = sum[i - 1] + I [i] cand = sum[i] - min M = MAX{M, cand} min = MIN{min, sum[i]} 5: } 6: Output M

There are other algorithms for MSP in one-dimension. These are mentioned in this thesis, for example Smith's O(n) time algorithm for one-dimensional array and alternative O(n) time divide-and-conquer algorithm for one dimension. The reader can refer to [1] to learn more about the computation of MSP in one dimension.

2.1.2 Maximum Subarray in two-dimensional array

The maximum subarray in two-dimensional array can be computed for a given input array of size m × n (we assume that m ≤ n). Some one-dimensional array algorithms mentioned previously were extended to solve the two-dimensional array problems, such as Kadane's algorithm, maximum sum in a one-dimensional array, and Smith's O(n) time algorithm. The following sections cover Kadane's algorithm and the prefix sum using two-dimensional arrays.

2.1.2.1 Kadane's algorithm in two-dimensional array

The one-dimensional Kadane's algorithm is performed for the strip defined by row x and row z as shown in Figure 2.3 below. The rectangular shape defined by (x1,y1) and (x2,y2) at the bottom-right and the top-left corner is the tentative one for the solution, while the rectangular shape defined in the strip from l to k is a tentative one for this particular one-dimensional case defined by x and z. As we solve the one-dimensional Kadane by changing x and z, the computation time of this algorithm is $O(m^2 n)$ in total, and the value of column[x][i] is the sum of a[z..x][i], that is, the sum of the i-th column of array a[][] from row z to row x.

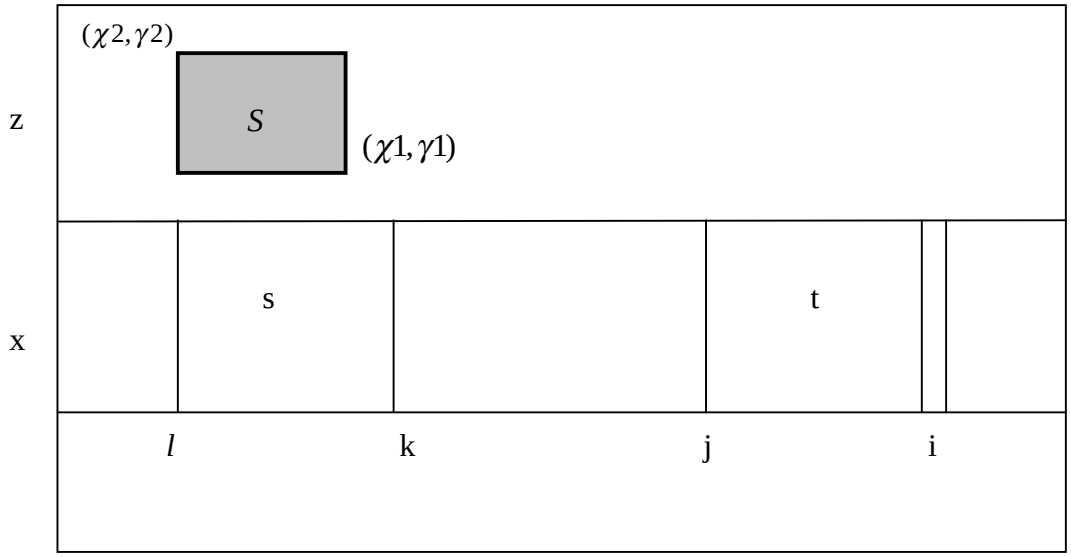


Figure 2. 3: An example to illustrate Kadane's algorithm in two-dimensional Array

Algorithm (4)
Kadane's two-dimensional algorithm

```
1.  ((x1,y1),(x2,y2)) = ((0,0),(0,0));
2.  S=0;
3.  for(z=1;z<=m;z++){
4.      /** initialize column[][]**/
5.      for(i=1;i<=n;i++) column[z-1][i]=0;
6.      for((x=z;x<=m;x++){
                    t=0;s=0;(k,l)=(0,0);
                     j=1
7.                    for(i=1;i<=n;i++){
                              column[x][i]= column[x-1][i] + a[x][i];
                               t = t + column[x][i];
                              if( t > s ){ s = t ;(k,l) = (i,j); }
                              if( t < 0){ t = 0;j = i+1;}
```

```
8.                    }
9.          if (s>S){S=s;x1=x;y1=k;x2=z;y2=l;}
10.     }
11. }
```

2.1.2.2 Algorithm for two-dimensional array by prefix sum

The prefix sum at position (i,j) of a two-dimensional array is the sum of array portion a[1..i][1..j]. Using the data structure of *column* given below, we have the following algorithm for array *sum* with O(mn) time, that is, linear time in the two dimensions

Algorithm (5)

Prefix sum algorithm O(mn)

```
    /** initialize column[][]**/
1.  for (j = 1;j ≤ n; i ++) column[0][j]=0;
2.  for(i=1;i≤m;i++){
3.      for (j=1;j≤n;j++){
            column[i][j]=column[i-1][j]+a[i][j];
            sum[i][j]=sum[i][j-1]+column[i][j];
4.      }
5.  }
```

Using the concept of prefix sum, we can develop algorithms for the maximum subarray problem, as will be shown in the following subsection.

2.1.2.2.1 Algorithm for one-dimensional array case : alternative to Kadane

In algorithm (6), *min_prefix* is the minimum prefix sum at the end of iteration *i*, and k_{min} holds the position for it. The variables holds the maximum subarray value in the portion [1 .. i] at the end of the i-th iteration. sum[i]-min_prefix is the maximum sum ending at position i. If this is greater than s, s must be replaced. If sum[i]<min_prefix, min_prefix must be replaced.

Algorithm (6)
Prefix sum algorithm

1. min_prefix = 0; s = -999; k = 0; l = 0; k_{min} = 0;
2. for (i=1;i ≤ n; i++){
if(sum[i]-min_prefix>s){s=sum[i]-min_prefix;l =I;k=k_{min};}
if (sum[i]<min_prefix){min_prefix=sum[i];k_{min}=i;}
3. }

Figure 2.4 illustrates the computation of Algorithm (6).

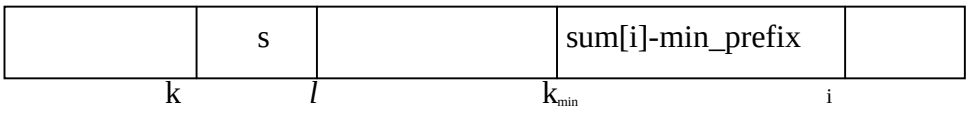


Figure 2. 4: The computation of Algorithm (6)

2.1.2.2.2 Algorithm for two-dimensional array case

In Algorithm (7), the 1D problem is solved repeatedly for the strip bounded by row z and row x in a similar manner to the two dimensional Kadane.

Algorithm (7)

Prefix sum 2 two-dimensional array algorithm

```
1.  ((x1,y1),(x2,y2)) = ((0,0),(0,0));
2.  S=0;
3.  for (z=1;z≤m;z++){
        for(x=z; x≤m,x++){
            t=0;s=0;(k,l)=(0,0); k_min =0;min_prefix=0;
            for (i=1;i ≤ n; i++){
                t=sum[x][i]-sum[z-1][i]-min_prefix;
                if(t > t) { s = t; k = k_min; l = i }
                if (sum[x][i]-sum[z-1][i] < min_prefix){
                    min_prefix=sum[x][i]-sum[z-1][i];
                    k_min =i;  }
4.          }
5.  if(s>S){S=s;x1=x;y1=l;x2=z;y2=k+1;}
6.     }
7.  }
```

An example of the prefix sum for a two-dimensional array is given in Figure 2.5.

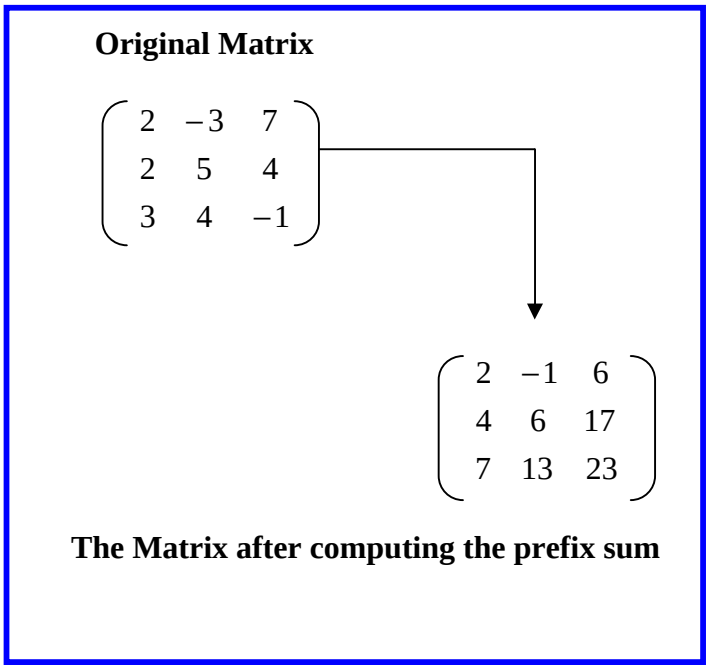


Figure 2. 5: An example on the prefix sum in two-dimensional array

Besides the extended algorithms, a framework specific to two-dimensional arrays based on distance matrix multiplication (DMM) is known [20]. Furthermore, using O(n) time algorithm for one dimensional, Smith derived a two-dimensional array algorithm that achieves $O(m^2 n)$ time, by using the divide-and-conquer framework. It was also found by Tamaki and Tokuyama that a divide-and-conquer framework similar to Smith's can be related to Takaoka's distance matrix multiplication (DMM) [20], which reduced the upper bound to sub-cubic time. Takaoka later developed a simplified algorithm with the same complexity [8].

2.2 K-maximum subarray problem

Many applications require finding multiple maximum subarrays; that is, finding the

first, the second, third…., and k^{th} maximum sum. There are two types of the K maximum subarray problem, K Disjoint MSP and K Overlapping MSP. In the following subsections, the two types are going to be discussed.

2.2.1 Disjoint k-maximum subarry problem

The disjoint k-maximum subbarray problem starts by taking the remaining portion of the array. In other words, after discarding the first maximum sum portion, the remaining array gets processed in the algorithm. This procedure of finding the K^{th} maximum sum continues for the second, third, …., K^{th} maximum sum.

The algorithm for finding the K^{th} disjoint maximum sum problem can be implemented as follows (there are many algorithms for finding the maximum sum; any one of these could be used): after finding the first maximum sum, the elements of the found portion of the maximum sum are given the value of minus infinity (-999). When the second maximum sum is found, the elements of the array portion corresponding to that are replaced by minus infinity (-999). This process continues for the third, fourth,…., K^{th} maximum sum. This approach takes O(kn) time for the one dimensional case, and can be extended to two dimensions. This results in O ($km^2 n$) time.

The above procedure for finding the K^{th} disjoint maximum sum can be used in many applications. One crucial application is finding brain cancer tumours. A brief

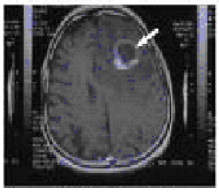
introduction to brain cancer is given in this section, to illustrate how finding the K^{th} maximum sum can be used in this particular field.

A brain tumour is the abnormal growth of a mass of unnecessary cells in the brain. There are two types of brain tumour, "benign" and "malignant" tumours. There is further classification of brain tumours into primary and metastatic. A primary tumour starts in the brain, while a tumour that starts elsewhere in the body and then travels to the brain is called metastatic.

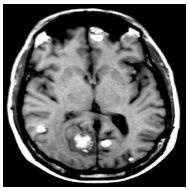
The primary brain tumour is benign since it rarely spreads, grows slowly, and has distinct boundaries. The treatment of this type is effective. However, if it is located on a vital area of the brain, it is considered life threatening.

The metastatic type of brain tumours is malignant, since they begin as cancer elsewhere in the body and spread to other areas. They vary in the way they grow and how they respond to treatment. One type of malignant tumour is encapsulated and relatively easy to remove. Others have long, thin filaments spreading through the brain, like roots of a plant.

The K^{th} disjoint maximum sum algorithm can be applied in the case of brain cancer to find tumours; that is, to find the first, second, third,…., until all affected areas are diagnosed (Figure 2.6 shows an example of brain cancer tumour). This process is called the K disjoint maximum subarray problem.



Brain cancer cell



Multiple brain cancer cells

Figure 2. 6: This figure shows affected areas of the brain by cancer cells

2.2.2 K Overlapping maximum subarry problem

Maximum subarray problem was extended to K-MSP by Bae and Takaoka [1]. This is an extension of the original problem of MSP, with finding maxima by only changing coordinates slightly from the found portion. Instead of searching for only one maximum subarray we search for K maximum subarrays. For example, in Figure 2.7 we identified the first and second maximum sum by the overlapping technique. S maximum sum is one of the maxima before finding the K^{th} maximum sum.

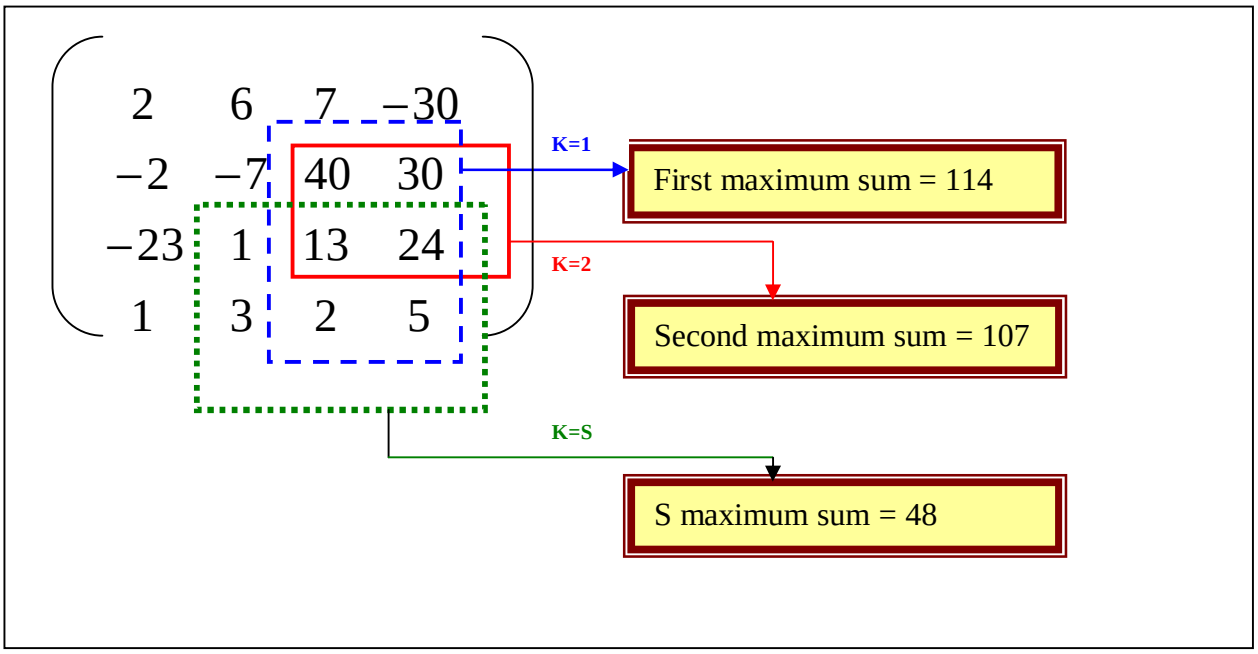


Figure 2. 7: An example to show the overlapping MSP

2.3 Maximum convex sum problem

Research done previously and current methods of finding MSP include using the rectangular shape for finding maximum sum or gain. The rectangular shape region used previously is not flexible enough to cover various data distributions. This research suggested using the convex shape, with the expected result of efficient and optimized outcomes. The convex shape is more likely to cover a wider range of data distributions.

The convex shape can be defined as follows: suppose that we have two lines crossing each other right angles, horizontal and vertical lines running along the x-axis and y-axis respectively. The convex shape is represented by points that belong to the shape (these points in the convex shape are connected along the lines continuously). Figure 2.8 shows how points can be connected.

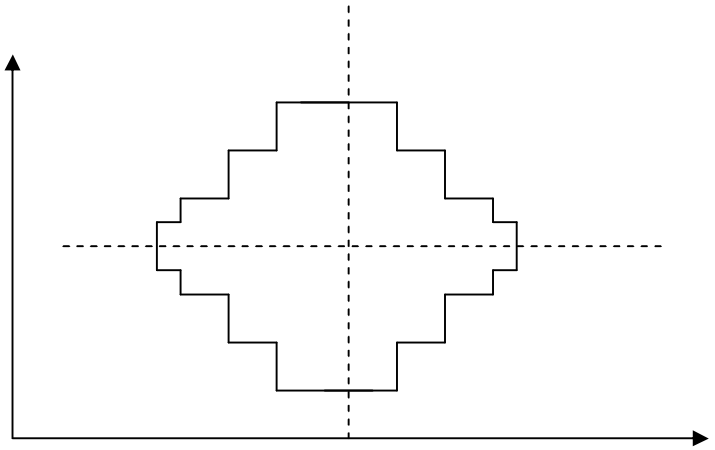


Figure 2. 8: A convex shape that can be used to find the maximum sum

An example of using a convex shape is shown in Figure 2.9, where data was obtained from bank records. The rectangular shape is not flexible enough when compared with the convex shape in Figure 2.9, where the red colour (darker colour) shapes are targets that help in obtaining a maximum gain or sum.

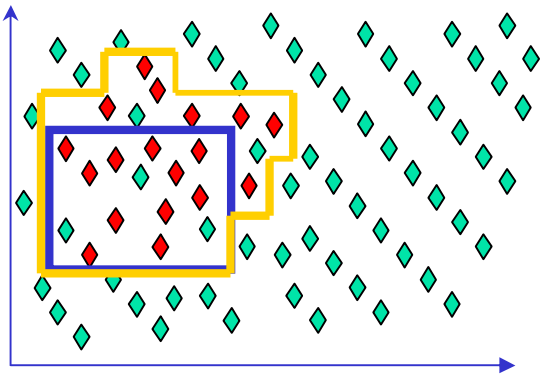


Figure 2. 9: This figure shows data obtained from bank records. The Rectangular shape and the convex shape were used to find the maximum sum of the data. The convex shape is flexible compared to the rectangular shape and includes more data distributions.

2.3.1 Definition of the convex shape based on the monotone concept

Definition 2.3.1.1: *x-monotone(curve)*

A curve is *x-monotone* if any vertical line either does not intersect, or it intersects in a single point.

Example:

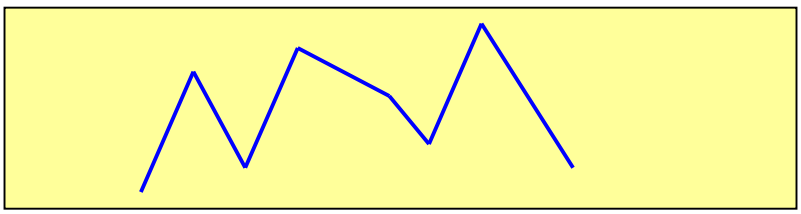


Figure 2. 10: *x-monotone* (curve)

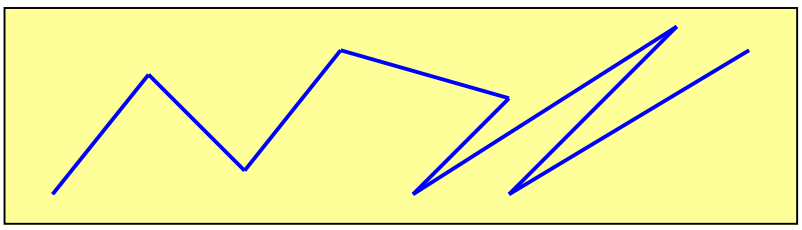


Figure 2. 11: n*on x-monotone* curve

Definition 2.3.1.2: *acceptable region*

Any connected region in the two-dimensional Euclidean space denoted $\mathbb{R}^2$ is called an acceptable region, which is acceptable in terms of research purposes. See figure 2.12

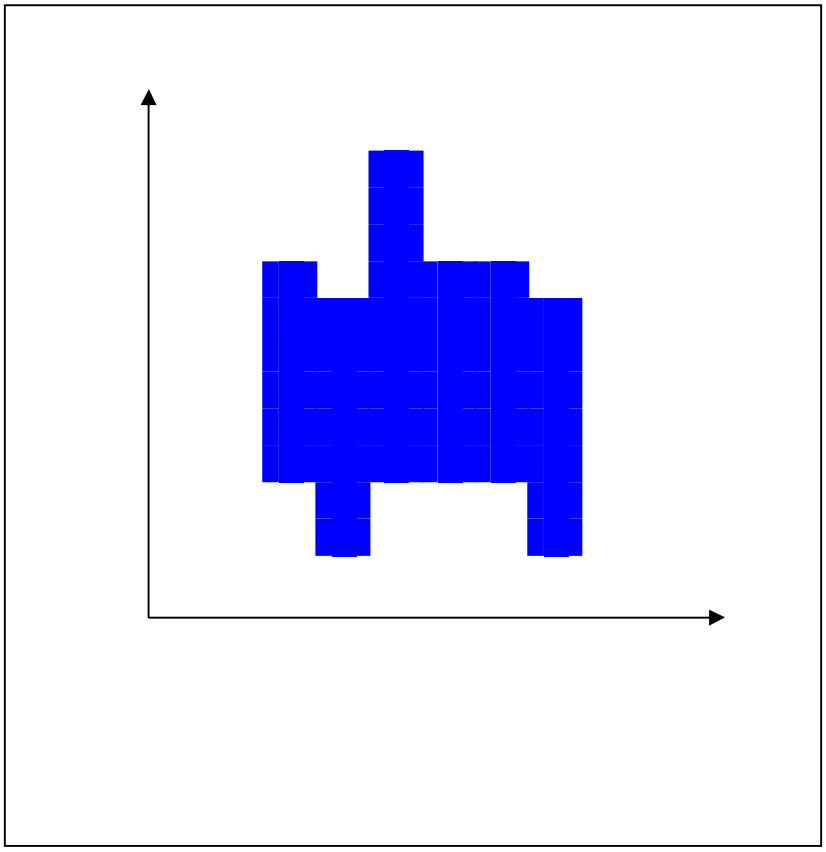


Figure 2. 12: A figure to show an example of an acceptable region

In the connected shape we consider two classes of geometric regions: rectangular regions and acceptable regions; more details will be covered in chapter three.

Definition 2.3.1.3: *x-monotone(region)*

A region is called an *x-monotone* region if the connected shape intersected by any vertical line is undivided.

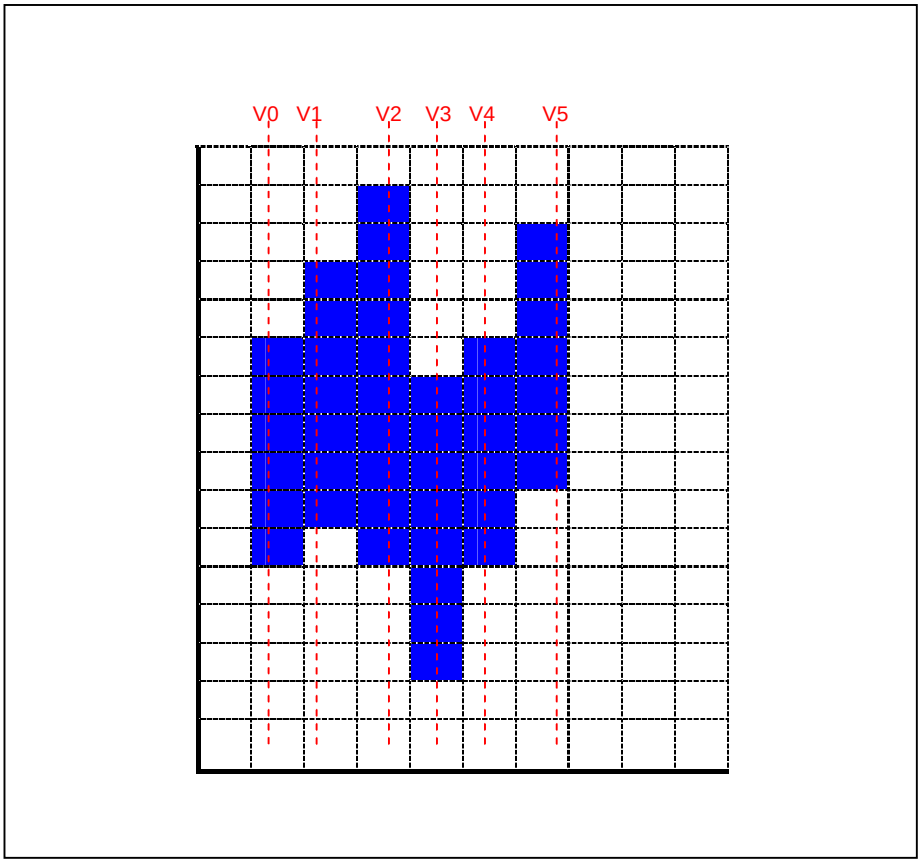


Figure 2. 13: *x- monotone* region

To illustrate the *x-monotone* region definition in a mathematical manner, suppose that we have a shape and the set *S* is parallel lines (these parallel lines containing a set of points are connected along the lines continuously*)*, *S*= {V0, V1, V_2, V_3, V_4, V_5, V_6, $V_{7,.....}$n}.The *x-monotone* shape can be defined as the undivided parallel lines of set *S* located inside the connected shape.

Definition 2.3.1.4: *y-monotone (region)*

A region is called a *y-monotone* region if the connected shape intersected with any horizontal line is undivided .

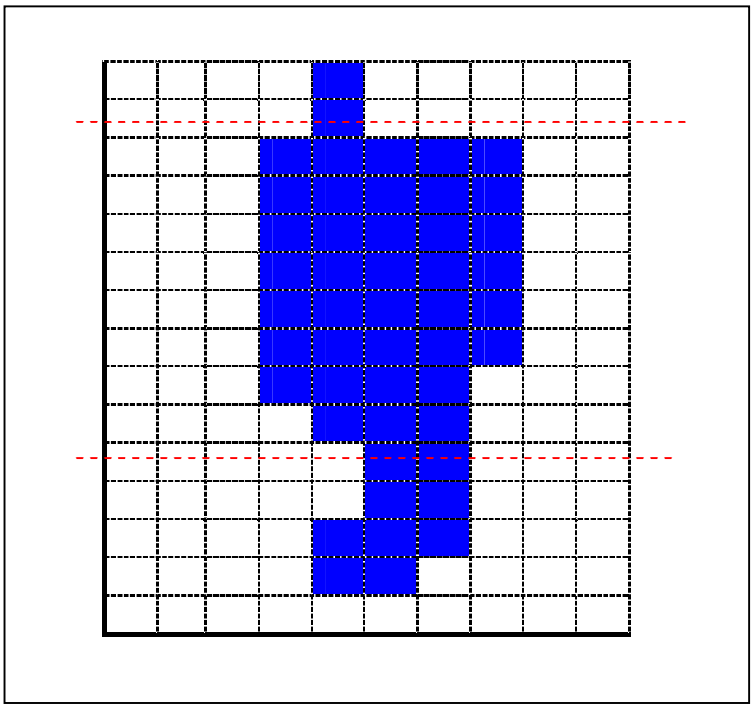


Figure 2. 14: *y-monotone* region

Definition 2.3.1.5: *the convex shape*

The convex shape figure 2.15 can be defined as the acceptable region, That is both *x-monotone* and *y-monotone*.

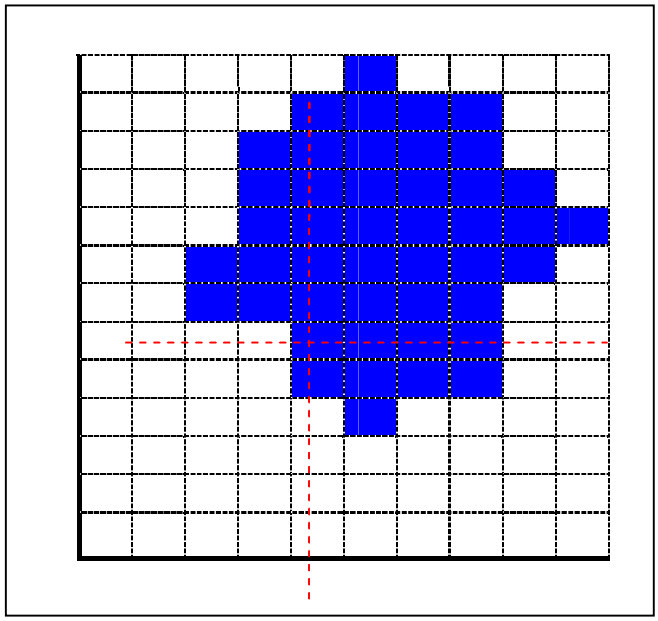


Figure 2. 15: Convex shape (*x-monotone* region and *y-monotone* region)

2.4 Chapter Summary

Two classes of geometric regions are considered in the connected shape: the rectangular shape and convex shape. Chapter two covered MSP in one-dimension and two-dimensions based on the rectangular region which was covered in detail, including the prefix sum algorithms, and K-maximum algorithms. Also, this chapter briefly covered the maximum convex sum problem. A detailed section on the new approach will be covered in chapter three.

Chapter 3

Convex Shape

In this chapter, the convex shape is covered in detail. The chapter starts with a section on the Venn diagram related to set of shapes used in this thesis. Proof of relevant results to the convex shape and MSP are presented in this chapter. Also, in this chapter the two main recent results that were derived/achieved are discussed; the first finding/derivation uses the convex shape in the maximum sum problem (MSP), and the second is the derivation of the K^{th} maximum sum for the convex shape by using the disjoint technique, which can be used in many applications. This chapter also presents a section on the efficiency, which was achieved by the following: the highly selective aspect of the algorithm, determining the boundaries of the shape, and minimising the speed of the algorithm of the convex shape by using the prefix sum method. Finally, a section on the K convex shape algorithm is covered.

3.1 Venn Diagram

A Venn diagram was derived based on the definitions of x-monotone shape, y-monotone shape, convex shape, and rectangular shape in the MSP context (these definitions were covered in detail in chapter 2).The Venn diagram in Figure 3.1 has a

domain of any connected shape, for example a triangle. The sets of the Venn diagram show the x-monotone (left/green), the y-monotone (right/red), and the intersection region includes a convex shape and included in the convex shape region is the rectangular shape. The area of intersection that represents the convex shape indicates that it has unique characteristics of all shapes, since it covers various data distributions because of its flexibility.

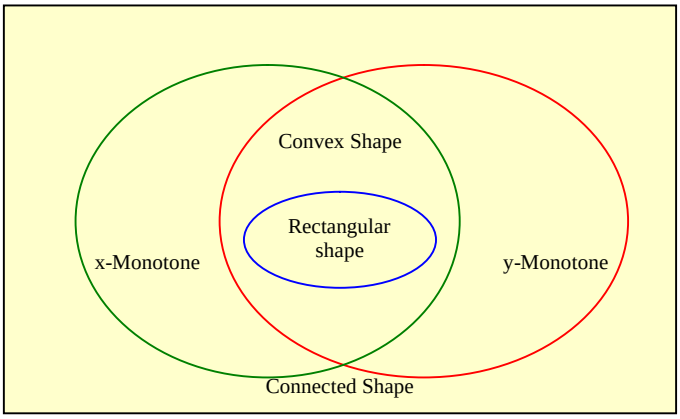


Figure 3. 1: The Venn diagram of shapes used in MSP with a domain of any connected shape and sets of x-monotone and y-monotone. The area of intersection has the convex shape region.

3.2 The convex shape algorithm

There are very few papers that discuss the convex shape. One of these papers is written by Fukuda and Morimoto (the reader can refer to [7] for more details).

In this research, the convex shape and implementing the related algorithms is studied in depth with the focus on a particular type from the convex shapes family. This type is shown in Figure 3.2, which is called the convex WN shape. This shape, as will be seen, has unique characteristics that supports its varied applicability. One such

application is the use of the convex shape to accurately locate cancer lumps in brain tumours. This finding was achieved based on a study done by the researcher with the cooperation of Christchurch hospital in New Zealand; this was mentioned in Chapter 1 in the medical application section.

3.2.1 Implementation of the convex (WN) shape

3.2.1.1 Definition: W Shape

A region that gets wider from left to right is named W, where the top of each y-column should be at least as high as the previous column, and the bottom of each y-column should be at least as low as the previous.

3.2.1.2 Definition: N Shape

A region that gets narrower from left to right is named N, where the top of each y-column should be at least as low as the previous column, and the bottom of each y-column should be at least as high as the previous.

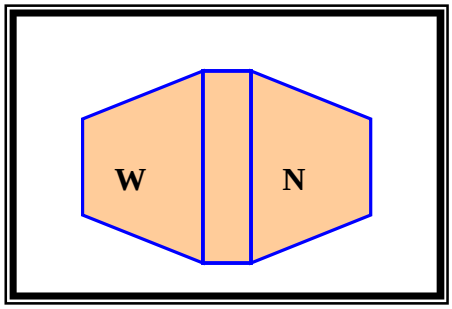


Figure 3. 2: The Convex (WN) shape

3.2.1.3 Theorem

The Maximum sum for the convex shape can be computed in O (n^3) time (refer to 3.2.1.5 for proof).

.

3.2.1.4 Detailed definition of W shape

The algorithm used in finding the W shape, finds all the W shape possibilities in two-dimensions, based on dynamic programming. There are three scenarios in step with

dynamic programming for finding the appropriate W shape in two-dimensions that returns the maximum sum. These scenarios are as follows: the first scenario (case 1) is based on the prevoius solution up to the (k-1)-th column. We need to check if the extension to the k^{th} column is approprite to maximise the gain or sum (Figure 3.3).

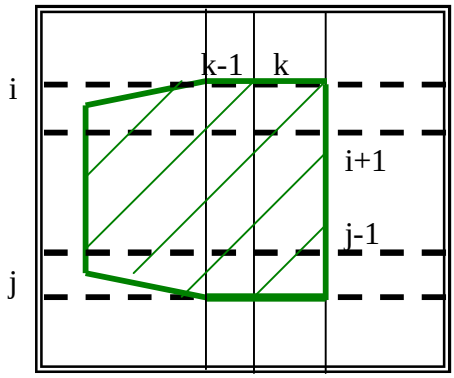


Figure 3. 3: First scenario of W shape

The second scenario (case 2) is the W shape area in Figure 3.4. Based on the previous solution in the k^{th} column with the narrower interval, we add the value at (j, k) to the best solution to get more gain or sum.

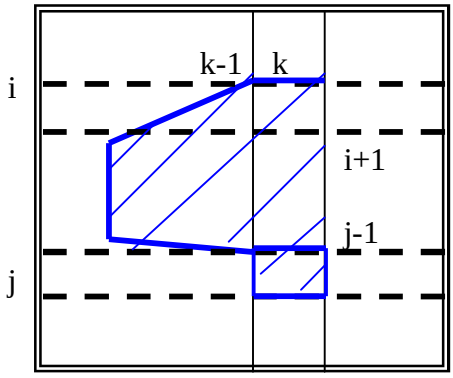


Figure 3. 4: Second scenario of W shape

The third scenario (case 3) is W shape with the narrower interval; we add the value at (i, k) to the best solution in the k^{th} column to get more gain or sum (Figure 3.5).

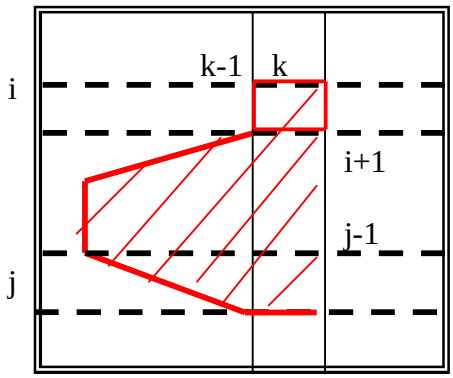


Figure 3. 5: Third scenario of W shape

To obtain the optimised solution using the W shape, the algorithm selects the maximum value of obtained maximum gain of the three scenarios mentioned above.

Let *a* be the array containing the original data. Let sum[k, s, t] be the sum of the k^{th} column from position s to t in the given two-dimensional array. In algorithm terms, f_W is computed as follows:

f_W (0, [s,t])=0 for all s < t
for k=1 to n do
  for all intervals of [s, t] in increasing order of t-s where s<t do

$$f_W(k, [s,t]) = \max \begin{cases} f_W(k-1, [s,t]) + sum[k, s, t] & \text{(case 1)} \\ f_W(k, [s+1,t]) + a[s, k] & \text{(case 2)} \\ f_W(k, [s, t-1]) + a[t, k] & \text{(case 3)} \end{cases}$$

Where f_W is the function to fine W shape. f_W (k,[s,t]) is the optimal value of the sum of W shape ending from position s to position t in column k. sum[k,s,t] is the sum of column k^{th} from position s to position t used in scenario one .a[s,k] and a[t,k] are the values added in scenarios two and three respectively.

To compute the boundaries of the convex shape, we keep track of the case number of the scenario that was taken in the above max-operation. Let array bw store the values of the case number, 1, 2, or 3 (bw array is going to be discussed in 3.4 section).

The above showed how to compute f_W. The values of f_N and bn are computed similarly for the N-shape. We simplified the algorithm by using the symmetry concept. The actual view perception of the shapes is symmetrical but the actual values for both W and N shapes are unequal. The WN shape (Figure 3.2) can be obtained, if the computation was performed bi-directionally and by merging the results. The proof of this is covered in detail in next section. This result discards steps that were previously considered important (after this realisation these steps became redundant). This gives optimised results.

3.2.1.5 Mathematical proof of the simplified algorithm using bidirectional computation

The convex shape can be found by the following (the detailed algorithm in section 3.3 -Algorithm 8- is written in C language).

The following algorithm is based on the results obtained from section 3.2.1.4 (Definition of W shape):

Let g(k, [s,t]) be the column sum from s to t of the k-th column.

1. Compute W shape from left to right for each k, s and t in f_W.
2. Compute W shape from right to left for each k, s and t, resulting in f_N.
3. For k=1 to n do

   For s=1 to n do

   For t=s to n do

f_{WN} (k,[s,t])= (f_W (k,[s,t]) + f_N (k,[s,t] – g(k,[s,t])

4. Take the maximum of f_{WN} (k,[s,t]) for all k, s, t

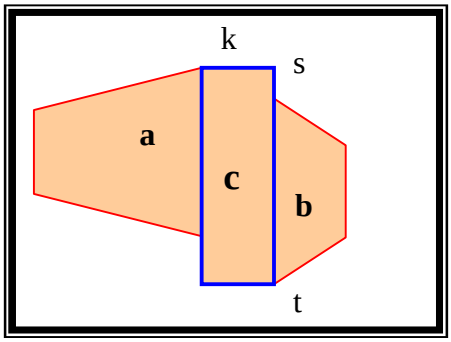


Figure 3. 6: This figure shows the Convex (WN) Shape presented in the proof

The following can prove the bidirectional characteristics in the convex shape case:

1. f_W =a + c maximum W
2. f_N = b + c maximum N
3. f_{WN} = a + b +c

Suppose f_{WN} is not maximum, then there must be another shape with the sum x + y + c > a + b + c, refer to the figure below (Figure 3.7)

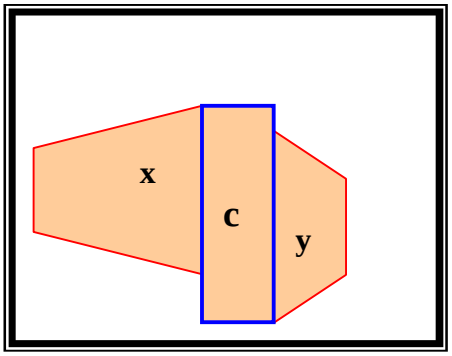


Figure 3. 7: The convex (WN) Shape used in the proof

That is x + y > a + b

Then x > a or y > b. otherwise x ≤ a and y ≤ b contradiction.

If x>a, x + c > a + c, contradicts with (1)

If y>b, contradicts with (2)

Proved

3.3 Convex Shape (WN) algorithm by using the prefix sum

Algorithm 8

Convex(WN) Shape algorithm by using the prefix sum O(n^3)

1. Read rows and columns in a matrix

2. Initializations

3. Compute prefix sum

```
for(i=1; i<=rows; i++)
    for(j=1; j<=cols; j++){
        pref[0][j]=0;
          for(k=1; k<=i; k++)
              pref[k][j]=pref[k-1][j]+a[k][j];
        }
```

4.W shape Calculation

```
for(k=1;k<=cols;k++){
    for(t=0; t<=rows-1;t++){
        for(i=1; i<=rows-t; i++){
            j=i+t; fw(i,j,k);
            };
        };
    };
```

5. N Shape Calculation

```
for(k=cols;k>=1;k--){
    for(t=0; t<=rows-1;t++){
        for(i=1; i<=rows-t; i++){
            j=i+t; fn(i,j,k);
            };
        };
    };
```

6.Convex (WN) Shape Calculation

```
for(k=1;k<=cols;k++){
    for(t=0; t<=rows-1;t++){
        for(i=1; i<=rows-t; i++){
            j=i+t;
            sum=pref[j][k]-pref[i-1][k];
            wn=aw[i][j][k]+an[i][j][k]-sum;
            if (wn>=maxwn){ maxwn = wn; im=i; jm=j;km=k;}
            };
        };
    };
```

7.Maximum Convex shape is im,jm,km,maxwn

8.Recording the boundaries of the (W) shape

```
kk=km; ii=im; jj=jm;
Record position (im,km) and (jm,km) in the region
```

```
while ((km>0)&&(aw[im][jm][km]>0)){
    {int i; for(i=im;i<=jm;i++)region[i][km]=1;}
    if(bw[im][jm][km]==1)km--; else
    if(bw[im][jm][km]==2)im++; else
    if(bw[im][jm][km]==3)jm--;
  }
 if((aw[im][jm][km]==0)&&(bw[im-1][jm][km]==2))
   region[im][km]=0;
 if((aw[im][jm][km]==0)&&(bw[im][jm+1][km]==3))
    region[jm][km]=0;
```

9. Recording the boundaries of the (N) shape

```
im=ii; jm=jj; km=kk;
while((km<=cols)&&(an[im][jm][km]>0)){
   { int i; for(i=im;i<=jm;i++)region[i][km]=1;}
    if(bn[im][jm][km]==1)km++; else
    if(bn[im][jm][km]==2)im++; else
    if(bn[im][jm][km]==3)jm--;
  }
 if((aw[im][jm][km]==0)&&(bw[im-1][jm][km]==2))
     region[im][km]=0;
 if((aw[im][jm][km]==0)&&(bw[im][jm+1][km]==3))
      region[jm][km]=0;
```

10. The boundaries of the convex (WN) shape

```
1.  for(i=1;i<=rows; i++){
    output i
      for(j=1;j<=cols;j++)printf("%3d ", region[i][j]); printf("\n");
    }
```

11. Output original matrix

```
original_a(int j, int k){
  int ii,jj;

      for(ii=1;ii<=rows; ii++){

          print  ii;

          for(jj=1;jj<=cols;jj++)printf("%3d ", a[ii][jj]);

      }

   }
```

1. Computing the function fw to find the W shape:

```
Initializations

if (k<=0){aw[i][j][k]=0;}

if (j<i) {aw[i][j][k]=0;}

Initialize fw(1,1,1), fw(2,2,1), fw(3,3,1) etc.

 if(k==1 && i==j) aw[i][j][k]=a[i][k];

     sum=pref[j][k]-pref[i-1][k];

       f1=aw[i][j][k-1]+sum;

       f2=aw[i+1][j][k]+a[i][k];

       f3=aw[i][j-1][k]+a[j][k];

 /* return the maximum of f1,f2 and f3*/

   if (f1>=f2 && f1>=f3) { aw[i][j][k]=f1; bw[i][j][k]=1; }

        else{ if(f2>=f1 && f2>=f3){ aw[i][j][k]=f2; bw[i][j][k]=2; }

            else { aw[i][j][k]=f3; bw[i][j][k]=3; }

          }
```

2. Computing the function fn to find the N shape:

```
Initializations

if (k>=cols)an[i][j][k]=0;

if (j<i) an[i][j][k]=0;

initialize fn(1,1,n), fn(2,2,n), fn(3,3,n) etc.
```

```
if(k==cols-1 && i==j) an[i][j][k]=a[i][k];
    sum=pref[j][k]-pref[i-1][k];
f1=an[i][j][k+1]+sum;
f2=an[i+1][j][k] + a[i][k];;
f3=an[i][j-1][k]+a[j][k];
return the maximum of f1,f2 and f3
if (f1>=f2 && f1>=f3) {an[i][j][k]=f1; bn[i][j][k]=1;}
      else{if(f2>=f1 && f2>=f3) {an[i][j][k]=f2; bn[i][j][k]=2;}
          else {an[i][j][k]=f3; bn[i][j][k]=3;}
       }
```

Example 3.3.1: if we are given a two-dimensional array a [1..m][1..n], suppose the upper-left corner has coordinates (1,1). The maximum subarray in the following example, when running any one of the MSP algorithm in two-dimensional array such as Kadane's algorithm, maximum sum in a two-dimensional array or Smith's O(n) time algorithm, the maximum sum in the 2D array below will be located in the array portion a[3..4][5..6] surrounded by inner brackets, which has the sum of 15, The time complexity is $O(n^3)$

$$A = \begin{pmatrix} -1 & 2 & -3 & 5 & -4 & -8 & 3 & -3 \\ 2 & -4 & -6 & -8 & 2 & -5 & 4 & 1 \\ 3 & -2 & 9 & -9 & -1 & 10 & -5 & 2 \\ 1 & -3 & 5 & -7 & 8 & -2 & 2 & -6 \end{pmatrix}$$

Figure 3. 8: Maximum sum obtained from using the rectangular shape

Using the same example, if the convex shape of Algorithm 8 is run to find the maximum sum, the returned result is a maximum gain of 23 (Figure 3.9). This solution is an optimised result compared to that obtained using the rectangular shape algorithm. Detailed examples using larger matrices are covered in chapter 4.

$$A = \begin{pmatrix} -1 & 2 & -3 & 5 & -4 & -8 & 3 & -3 \\ 2 & -4 & -6 & -8 & 2 & -5 & 4 & 1 \\ 3 & -2 & 9 & -9 & -1 & 10 & -5 & 2 \\ 1 & -3 & 5 & -7 & 8 & -2 & 2 & -6 \end{pmatrix}$$

Figure 3. 9: Maximum sum obtained from using the convex shape

3.4. Finding the maximum sum while determining the boundaries of the convex shape

This research presents a significant result by determining the actual convex shape used in finding the maximum sum (Fukuda and Morimoto's previous study has not practically given the boundaries of the convex shape). The boundaries of this shape were determined by Algorithm 8 to find the actual shape. This result was achieved by maintaining the information in an array, for example bw in Algorithm 8 depends on which direction the optimal values were obtained in the equation that were discussed in section 3.2.1.5. The array bn is similar for the N shape. These arrays are used to obtain the actual shape for the maximum sum value.

A brief sketch of the above mentioned algorithm is as follows: let "region" be an (m, n)-array to identify the shape. It is initialized to 0. In the following programme, for each column k, the upper boundary and the lower boundary are set to 1. We start from the indices obtained from step 4 in section 3.2.1.5 (convex shape algorithm) which is (Take the maximum of f_{WN} (k,[s,t]) for all k, s, t) . This backtracks the computation of the WN shape. In similar manner to the above algorithm, the N shape can be computed.

```
while (k>0 and f_W(k, [s, t])>0){
    region[s][k]=1;
    region[t][k]=1;
    if(bw[s][t][k] =1)  k--;
    else  if(bw[s][t][k]=2)  i++;
    else  if(bw[s][t][k]=3) j--;
 }
```

3.5 Increasing the efficiency of the convex shape algorithm

This research presents an efficient algorithm, which uses the convex shape to find the maximum sum. The efficiency of this algorithm comes about as a result of the following: the first aspect is that the algorithm is highly selective in choosing the acceptance region's boundaries. This algorithm finds the maximum sum and at the same time minimizes the size of the selected region. For example, consider number five in Figure 3.10, which is part of the overall shape. In the case of a non-efficient algorithm the adjacent numbers that result in a sum of zeros would be included in the summation process. In our case, where the algorithm is efficient in selecting the required region, these numbers are discarded and the boundary of the area stops at number five.

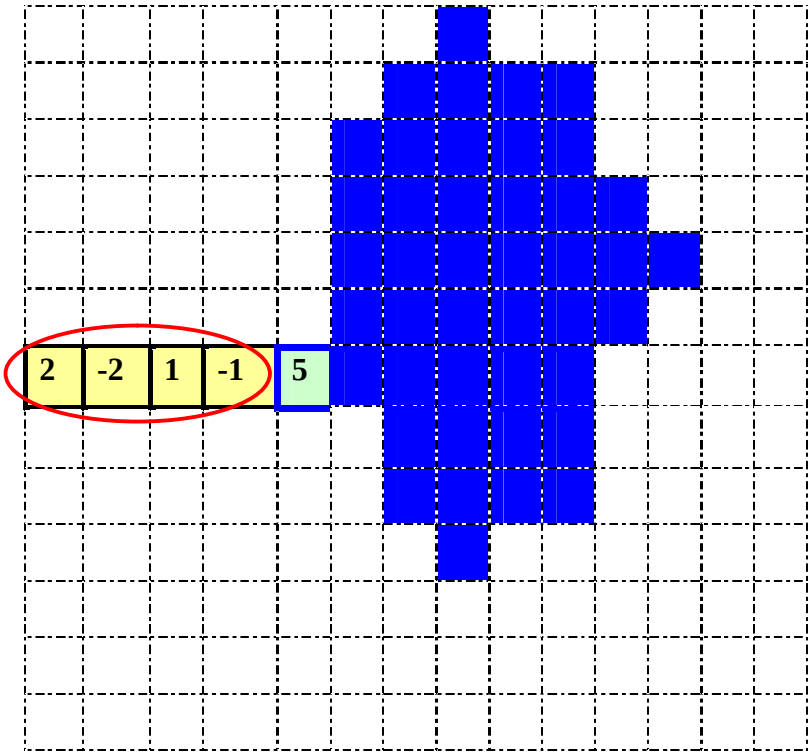
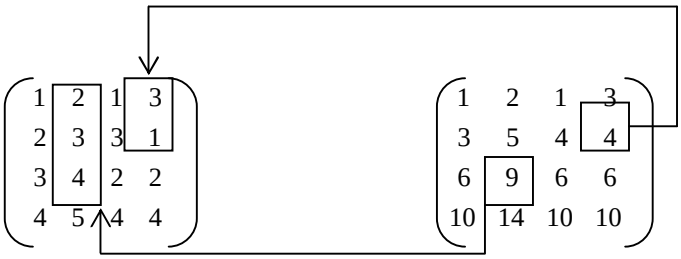


Figure 3. 10: An example that shows the selective aspect of the algorithm

The second merit of our approach is to increase the efficiency of the convex shape algorithm by running the prefix sum algorithm on columns of the matrix (Figure 3.11). This will prevent repeating the summation process over and over again for the elements. The following algorithm is an example that covers this aspect. The gain function is given by g (k,[s,t]) to be the sum of the k-th column from position s to position t. Let sum (k,s) be the prefix sum of column k from position 1 to position s. Then g (k,[s,t]) can be computed by g(k, [s,t]) = sum(k, t) – sum(k, s), after sum(k, s) are computed for all k and s in $O(n^2)$ time.

For example: g(2, [2,3]) = sum(2, 4) – sum(2, 2)

= 9-2 = 7.



Original Matrix **Output Matrix (using prefix sum in each column)**

Figure 3. 11: Prefix sum enhances the efficiency of the convex shape algorithm

3.6 K convex sum problem

In chapter 2, the maximum sum problem using the K^{th} sum was covered. In a similar manner, the K^{th} maximum sum using the convex shape is covered in this section. The K-disjoint approach is applied in finding the K^{th} convex shape.

3.6.1 Disjoint Convex Sum Problem (Algorithm 9)

The K-disjoint convex sum problem starts by taking the remaining portion of the array. This means after discarding the first maximum convex sum portion, the remaining array gets processed in the algorithm. This procedure of finding the K^{th} maximum convex sum continues for the second, third, …., K^{th} maximum convex sum.

The algorithm for finding the K^{th} disjoint maximum convex sum can be implemented as follows (Algorithm 9): the first maximum convex sum can be found by using Algorithm 8. After finding this, the elements of the found portion of the maximum convex sum are given the value of minus infinity (-999). When the second maximum convex sum is found, the elements of the array portion corresponding to this are

replaced by minus infinity (-999). This process continues for the third, fourth,…., K^{th} maximum convex sum. This approach takes O ($km^2 n$) time in 2D.

The above procedure for finding the K^{th} disjoint maximum convex sum can be used in many applications. One important application was mentioned in chapter 2, which is finding brain cancer tumours. In Figure 2.6 (chapter 2), the algorithm is run to find the first maximum sum, which corresponds to the first lump. Then, the second lump is found by running the algorithm to find the second maximum sum. This is repeated until all lumps in the brain are located.

Algorithm 9

K Convex shapes by using disjoint technique

```
1. Read rows and columns in a matrix
2. initializations
3. while(1){
            for(i=0;i<=rows;i++)
              for(j=i;j<=rows;j++)
                for(k=0;k<=cols;k++){
                     aw[i][j][k]=0; an[i][j][k]=0;
                      bw[i][j][k]=0; bn[i][j][k]=0;
                 }
4.Compute prefix sum
              for(i=1; i<=rows; i++)
                  for(j=1; j<=cols; j++){
                       pref[0][j]=0;
                       for(k=1; k<=i; k++)
                           pref[k][j]=pref[k-1][j]+a[k][j];
```

```
                }
```

5. W Shape calculation

```
        for(k=1;k<=cols;k++){
            for(t=0; t<=rows-1;t++){
                for(i=1; i<=rows-t; i++){
                    j=i+t; fw(i,j,k);
                };
            };
        };
```

6. N shape calculation

```
            for(k=cols;k>=1;k--){
                for(t=0; t<=rows-1;t++){
                    for(i=1; i<=rows-t; i++){
                    j=i+t; fn(i,j,k);
                    };
                };
            };
```

7. Convex (WN) calculation

```
            maxwn=-999;
            for(k=1;k<=cols;k++){
                for(t=0; t<=rows-1;t++){
                    for(i=1; i<=rows-t; i++){
                        j=i+t;
                        sum=pref[j][k]-pref[i-1][k];
                        wn=aw[i][j][k]+an[i][j][k]-sum;
                        if (wn>=maxwn){ maxwn = wn; im=i; jm=j; km=k;}
```

```
                    };
                };
            };

if(maxwn<0) post_process(im, km);
Print the results im,jm,km,maxwn;

8. Record position (im,km) and (jm,km) in the region
    kk=km; ii=im; jj=jm;
    while((km>0)&&(aw[im][jm][km]>0)){
                for(i=im;i<=jm;i++)region[i][km]=1;
                    if(bw[im][jm][km]==1)km--; else
                    if(bw[im][jm][km]==2)im++; else
                    if(bw[im][jm][km]==3)jm--;
      }
  if((aw[im][jm][km]==0)&&(bw[im-1][jm][km]==2))  region[im][km]=0;
  if((aw[im][jm][km]==0)&&(bw[im][jm+1][km]==3)) region[jm][km]=0;
    im=ii; jm=jj; km=kk;
   while((km<=cols)&&(an[im][jm][km]>0)){
           for(i=im;i<=jm;i++)region[i][km]=1;
                if(bn[im][jm][km]==1)km++; else
                if(bn[im][jm][km]==2)im++; else
                if(bn[im][jm][km]==3)jm--;
      }
    if((aw[im][jm][km]==0)&&(bw[im-1][jm][km]==2)) region[im][km]=0;
    if((aw[im][jm][km]==0)&&(bw[im][jm+1][km]==3)) region[jm][km]=0;
     for(i=1;i<=rows; i++){
             print  i;
```

```
            for(j=1;j<=cols;j++) print region[i][j] ;
        }


    for (i=1;i<=rows;i++)
    for(j=1;j<=cols;j++)
    if(region[i][j]==1)a[i][j]=-99;
} End while
post_process(int im, int jm){
    int i, j, max;
    do{
            print  a[im][jm],im,jm;
            max=-99;
            a[im][jm]=-99;
                for(i=1;i<=rows;i++)
                    for(j=1;j<=cols;j++)
                     if(a[i][j]>max){im=i; jm=j; max=a[i][j];}
      }
       while(max>-99);
            if(max=-99) exit(0);
  }
9.Output original matrix original_a(){
              int ii,jj;
              for(ii=1;ii<=rows; ii++){
                    print ii;
                   for(jj=1;jj<=cols;jj++) print a[ii][jj];
               }
    }
```

1. Computing the function fw to find the W shape:

```
Initializations
if (k<=0){aw[i][j][k]=0;}
if (j<i) {aw[i][j][k]=0;}
initialize fw(1,1,1), fw(2,2,1), fw(3,3,1) etc.
    if(k==1 && i==j) aw[i][j][k]=a[i][k];
       sum=pref[j][k]-pref[i-1][k];
    f1=aw[i][j][k-1]+sum;
    f2=aw[i+1][j][k]+a[i][k];
    f3=aw[i][j-1][k]+a[j][k];
 return the maximum of f1,f2 and f3
  if (f1>=f2 && f1>=f3) { aw[i][j][k]=f1; bw[i][j][k]=1; }
    else{ if(f2>=f1 && f2>=f3){ aw[i][j][k]=f2; bw[i][j][k]=2; }
        else { aw[i][j][k]=f3; bw[i][j][k]=3; }
```

2. Computing the function fn to find the N shape:

```
Initializations
if (k>=cols)an[i][j][k]=0;
if (j<i) an[i][j][k]=0;
 initialize fn(1,1,n), fn(2,2,n), fn(3,3,n) etc.
  if(k==cols-1 && i==j) an[i][j][k]=a[i][k];
     sum=pref[j][k]-pref[i-1][k];
    f1=an[i][j][k+1]+sum;
    f2=an[i+1][j][k] + a[i][k];;
    f3=an[i][j-1][k]+a[j][k];
   return the maximum of f1,f2 and f3
    if (f1>=f2 && f1>=f3) {an[i][j][k]=f1; bn[i][j][k]=1;}
```

```
else{  if(f2>=f1 && f2>=f3) {an[i][j][k]=f2; bn[i][j][k]=2;}
       else {an[i][j][k]=f3; bn[i][j][k]=3;}
    }
```

3.7 Chapter Summary

Chapter three covered the convex shape in detail. The convex shape algorithm and the K convex shape algorithm were developed to find the K convex shape by using the disjoint technique. This chapter showed how the efficiency of the algorithm was optimized by three factors; these are the highly selective aspect of the algorithm, boundary determining to define the actual shape, and using the prefix sum to speed up the algorithm. Chapter four presents the convex shape in more depth, showing experiments and evaluations.

Chapter 4

Evaluation

The convex shape plays a major role in the maximum sum problems (MSP) - previous research focuses on using the rectangular shape in MSP. The maximum convex sum problem opens up a new research area in MSP; its aim being to obtain precise outcomes and to optimise the solution. In this research, it was shown that the convex shape results in more optimised results compared to the rectangular shape. As explained, the rectangular shape was not flexible enough to cover various data distributions whereas the convex shape is. This implies that the gain is maximised by using the convex shape with the same time complexity. In this chapter evaluations of the algorithm outcomes are presented.

The implemented algorithms were done in C language. Our experiments were conducted on a Cyclone computer with GenuineIntel Dual Processors. Each processor has CPU 2.66GHz, cache size: 4096 KB, running under Linux Platform Fedora GNU GRUB Version 0.79. This chapter discusses the experimental part based on quantitative analysis. The experiment data were generated randomly by using a random number generator, which contains positive and negative values in two-dimensional array formed by using two digits in the range of 0 to 9.

Kadane's algorithm was implemented based on the rectangular shape region; section 2.1.2.1 in chapter 2 explains about this algorithm. The maximum sum generated data was found based on the rectangular shape. The new approach for dealing with MSP using the convex shape, which was covered in detail in chapter 3, was implemented. The challenge was to implement the convex shape in O (n^3) time complexity, which is the same for the rectangular shape (taking into account that finding the convex shape involves relatively complicated operations compared to the rectangular shape). Many techniques were used to speed up the convex shape algorithm, for example using the prefix sum, which is highly selective in choosing the acceptance region's boundaries.

4.1 Results and analysis

Table 3 shows the maximum sum or gain that was obtained from different matrices. The matrices sizes vary from 50 to 500. All the experiments were conducted on these matrices. It was observed that using the convex (WN) shape resulted in optimised outcomes. For example, when the algorithm is run on a 70x70 matrix, the resulted maximum sum outcome from using the rectangular shape was 4671, where the value of the maximum sum obtained from the convex shape was 13592.

	Max Sum	
Matrix Size(nxn)	**Rectangular Shape**	**Convex Shape**
50x50	2445	7000
70x70	4671	13592
100x100	7849	17958
150x150	15270	36247

200x200	14925	45551
250x250	22367	62300
300x300	22948	74686
350x350	19926	83544
400x400	23517	88062
500x500	35878	133514

Table 3: The maximum sum values obtained from applying the algorithms to different matrices

Figure 4.1 Shows the results represented in a columns chart; it presents the rectangular shape results and the convex shape results. As shown, the convex shape increases the maximum sum or gain. This implies that using the convex shape algorithm gives relatively precise results that can increase the benefits when used in different applications.

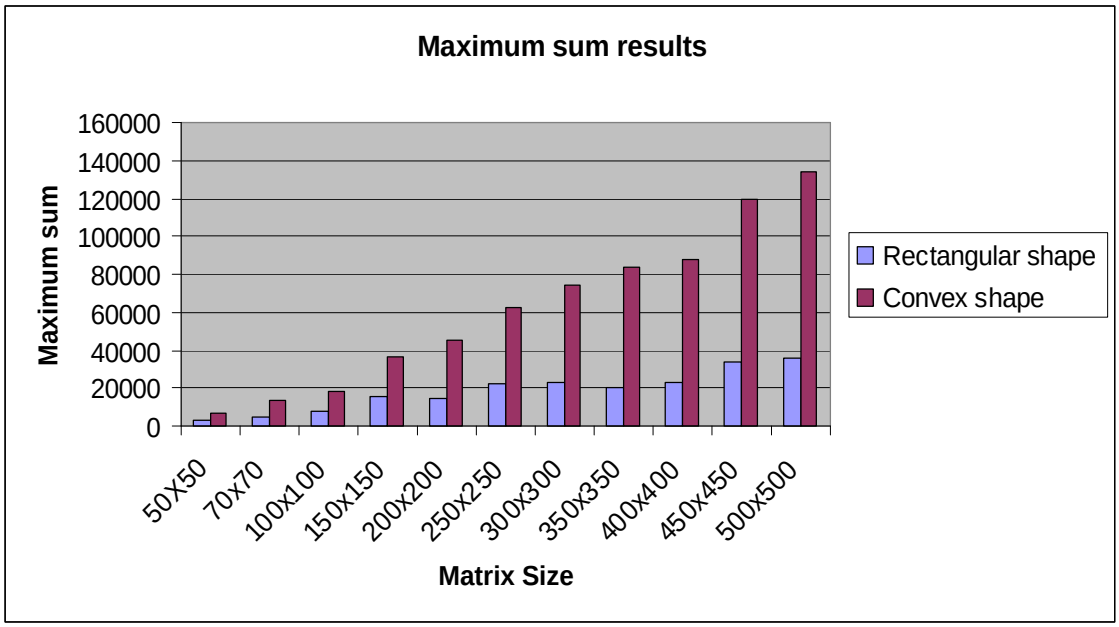


Figure 4. 1: Comparison columns of the maximum gain obtained from applying the convex shape and the rectangular shape algorithms.

Based on Table 3 and Figure 4.1 it can be concluded that the percentage of obtaining optimised results and maximum gain was exceeded when using the convex shape compared to the achieved percentages when using the rectangular shape (given that

when applying the algorithms, they were applied to the exact matrix and the percentage was then compared). Based on the conducted experiments, using the convex shape increases the percentage in the range of 60% – 80%. This indicates the flexibility of using the convex shape to cover various data, while achieving optimised results.

It is worthwhile in the discussion section to mention that there is still one possibility that the maximum sum obtained from using the convex shape equals that obtained from using the rectangular shape, when both shapes match in the Venn diagram.

The time complexity factor is important in any algorithm. This was achieved to be the same for both the convex shape and the rectangular shape, although the algorithm for the convex shape is more complicated and involves more operations compared to the rectangular shape algorithm.

Table 4 shows running time results that were obtained when running the two algorithms on the same matrices.

	Time measurement in sec.	
Matrix Size(nxn)	**Rectangular Shape**	**Convex Shape**
50x50	0	0
70x70	0	0.01
100x100	0.01	0.05
150x150	0.03	0.21
200x200	0.07	0.82
250x250	0.13	2.3
300x300	0.21	4.69

350x350	0.33	8.06
400x400	0.49	11.88
450x450	0.69	17.05
500x500	0.94	23.85

Table 4: The running time values obtained from applying the algorithms to different matrices

Figure 4.2 shows results obtained by measuring the running time for both the convex shape and the rectangular shape, The time complexity for the two algorithms is $O(n^3)$ Figure 4.2 indicates that the running time for the convex shape is relatively higher compared to the running time of the rectangular shape. This increase is the result of including more operations to find the maximum sum; for example the convex shape requires dealing with three- dimensional array, such that i, j and k indicate the location , where i and j determine the interval and k determines the current column. Even if this is the case, this algorithm is useful in terms of finding more precise and optimised results.

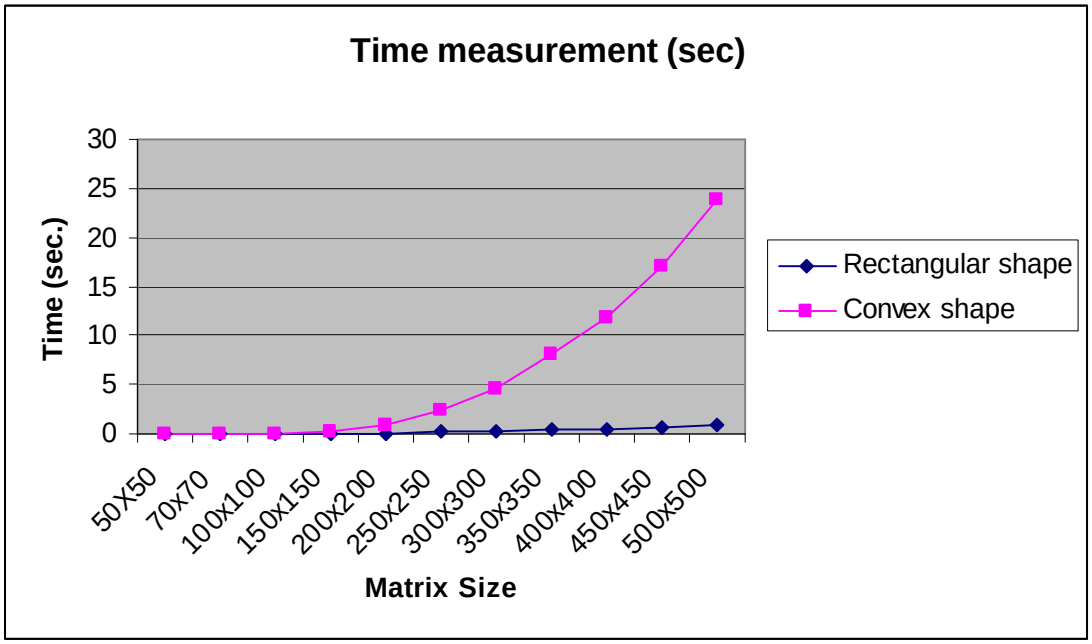


Figure 4. 2: The running time for the convex shape and the rectangular shape algorithms

4.2 Summary of chapter

Chapter four covered results, evaluations, and analysis of experiments that were conducted to compare the outcomes obtained from using the convex shape to that obtained by using the rectangular shape when applied to different matrices of different sizes. The results were tabulated and given in statistical charts.

It was proven that the results obtained from the convex shape algorithms give optimised maximum gain compared to the rectangular gain, while maintaining the same time complexity with an acceptable running time increase. This new approach of using the convex shape is more efficient and flexible and opens doors to the different potential applications, such as returning more revenue for the bank example mentioned in chapter one.

Chapter 5
Conclusion

This chapter summarises the work of the research and highlights the major findings. Two classes of geometric regions from the connected shapes family were considered in the research: the rectangular shape and the convex shape. It was proposed that the convex shape would increase the maximum gain/sum as compared to using the rectangular shape. Based on this hypothesis, the algorithm was generalised to include the K^{th} convex shape.

To prove the proposed objectives and to fulfill the research goals, the following steps were taken to reach the final outcomes of the research. The first step was to theoretically study the behaviour of both the rectangular shape and the convex shape. During this in-depth study, theories were proved by the researcher and theory guidelines were outlined. The second step was to implement an appropriate random number generator to start applying the theory to real number in the subsequent steps. The third step was to implement the convex shape, which was proven theoretically and mathematically before starting the coding phase. After implementing the convex shape, the rectangular shape was implemented. Along with implementing these algorithms, extra algorithms were required to serve the efficiency purposes of the project, such as implementing the prefix sum algorithm. Trial experiments were run on the implemented algorithms. Following this, the algorithms were edited to improve

the time frame and to reach the required stage, after which, the algorithm was generalised. This included finding the first, second, and subsequent sum up to the K^{th} maximum sum. The final stage was to apply the algorithms on different matrices that had different sizes, ranging from 50 to 500. The experiments were conducted to compare the results obtained from applying the convex shape and the rectangular shape algorithm.

In conclusion, it was found that using the convex shape algorithm resulted in optimised results compared to using the rectangular shape. This presents a new approach in applying MSP algorithms in real life applications. Before this research, the rectangular shape was confidently used in finding the maximum sum; it was assumed that it returned the best outcomes. This research over-rules this finding: the convex shape maximum gain is ensured to be optimised compared to existing algorithms maximum gain, which were based on the rectangular shape.

The results and findings of this research can be applied in many applications, such as in astronomy, medicine, graphics, economics and other disciplines that require finding relationships between variables and finding the peaks of the related data. This work of this research does not stop at the point where these objectives were achieved; it starts from there, where theory and practical algorithms were presented, and offers opportunities for further studies.

Future in-depth studies on the convex shape in MSP can be developed to include other options and advance the presented algorithms. Some of the areas that could be

used to further the research are to reduce the space requirement from n^3 to n^2, researching the K overlapping technique, and reducing the time complexity from $O(n^3)$ to sub cubic.

Bibliography

[1] - Sung Eun Bae, (2007). *Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem*, (pp 1- 150), University of Canterbury.

[2] Knuth, D. E. (1973). *The art of Computer Programming*, Vol. 3, Sorting and Searching, 2nd ed., Addison-Wesley (pp. 209-218).

[3] Tadao Takaoka, (2006). *Algorithms for Data Mining*, (pp.1-29), Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand.

[4] Takeshi Fukuda, Yasuhiko Morimoto and Takeshi Tokuyama, IBM Tokyo Research Laboratory. Shinichi Morishita, University of Tokyo, (June 2001). *Data Mining with Optimized Two-Dimensional Association Rules*, Vol. 26, No. 2, (pp. 179–213), ACM Transactions on Database Systems,

[5] International Telecommunication Union (1998.). Recommendation ITU-R BT. 470-6,7, Conventional Analog Television Systems.

[6] Srikant, R, and Agrawal, R. (1996), *Mining quantitative association rules in large relational tables,* (pp.1–12.) In Proc. of ACM SIGMOD International Conference on Management of Data.

[7] Fischer M , R. Miller and J. Thatcher, Eds. (1972) *Efficiency of equivalence algorithms*. *In Complexity of computer computations,*. (pp. 153–168) Plenum Press.

[8] FH Hochschule Eichmann, M. L¨ussi, Dr. Aggelos Katsaggelos and Dr. Guido M. Rapperswil. (December 2005) *Efficient Multilevel Image Thresholding*, Master degree.

[9] Arrays ,Dina Kravets and James K, (1994) *Selection and Sorting in totally Monotone* .Chapter 52.

[10] Takaoka, T., Voges, K., and Pope, N. (2005) *Algorithms for data mining*. *In Business Applications and Computational Intelligence*, (pp. 291–315) .Idea Group Publishing.

[11] Bentley, J. (1984), *Programming pearls: algorithm design techniques*, (pp. 865–873) Commun. ACM 27, 9.

[12] Takaoka, T. (2002), *Efficient Algorithms for the Maximum Subarray Problem by Distance Matrix Multiplication*. In Electronic Notes in Theoretical Computer Science, J. Harland, Ed., vol. 61, Elsevier.

[13] Bae, S. E. and Takaoka, T. (2006). *Improved Algorithms for the K Maximum Subarray Problem*. (pp. 358-374) The Computer Journal, Vol. 49, No. 3,.

[14] Bae, S. E. and Takaoka, T. (May 2004). *Algorithms for the Problem of K Maximum Sums and a VLSI Algorithm for the K Maximum Subarrays Problem*. (pp. 247–253) Proc. ISPAN 116 , Hong Kong,. IEEE Computer Society Press, Los Alamitos, CA.

[15] Wen. Z. (1995). *Fast Parallel Algorithms for the Maximum Sum Problem. Parallel Comput.*, (pp. 461-466) Vol. 21, No. 3,.

[16] Hannenhalli, S., and Levy, (2001). *Promoter prediction in the human genome. Bioinformatics* 17 S90–S96.

[17] Tamaki, H. and Tokuyama, T. (January 1998). *Algorithms for the Maximum Subarray Problem based on Matrix Multiplication*. (pp. 446-452) Proc. SODA, San Francisco, CA, 25-27,. SIAM, Philadelphia, PA.

[18] Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1975). *The Design and Analysis of Computer Algorithms*, (pp. 97-102), 2nd ed., Addison-Wesley,

[19] Tamaki, H., and Tokuyama, T., (1998), *Algorithms for the maximum subar-array problem based on matrix multiplication* (pp. 446–452), In Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms SIAM.

[20] Mohammad Bashar and Tadao Takaoka (2007), *Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem*, University of Canterbury.

[21] Lin, Y. L., Jiang, T., and Chao, K. M. (2002), *Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis*. (pp. 570–586), Journal of Computer and System Science 65.

[22] Cancer Society of New Zealand Inc, (2003), PO Box 10847,Wellington Copyright Third Edition 2005 ISBN 0-908933-65-7 Second Edition 2003 ISBN 0-9-08933-63-0 First Edition 1193 ,Reprinted 1994,1999,2000 ISBN 0-908933-05-3

[23] Cancer Society of New Zealand Inc, (2006) PO Box 10847, Wellington. First Edition 1994: ISBN 0-908933-18-5 Second Edition 2003: ISBN 0-908933-57-6 Third Edition: ISBN 0-908933-69-X

[24] The New Zealand Brest Cancer Foundation, (2006), Recourses http://www.nzbcf.org.nz/resources/, URL All Rights Reserved.

[25] Alan P. Sprague (October 29, 1999) *Extracting Optimal Association Rules over Numerical Attributes* (pp. 1–9) Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham Alabama 35294-1170